Low-Density Parity-Check Codes

Part I - Introduction and Overview

William Ryan, Professor
Electrical and Computer Engineering Department
The University of Arizona
Box 210104
Tucson, AZ 85721
ryan@ece.arizona.edu

2005

Low-Density Parity-Check Codes

- note that the parity-check matrix *H* is so called because it performs *m* = *n*-*k* separate parity checks on a received word $\overline{y} = \overline{c} + \overline{e}$

Example. With H^T as given above, the *n*-*k* = 3 parity checks implied by

$$\overline{y}H^T \overset{?}{=} \overline{0}$$

are

$$\begin{array}{ccccccccc} y_0 & + & y_1 & + & y_2 & + & y_4 & \overset{?}{=} & 0 \\ y_0 & + & y_1 & + & y_3 & + & y_5 & \overset{?}{=} & 0 \\ y_0 & + & y_2 & + & y_3 & + & y_6 & \overset{?}{=} & 0 \end{array}$$

Definition. A *low-density parity-check* (LDPC) *code* is a linear block code for which the parity-check matrix *H* has a low density of 1’s

Low-Density Parity-Check Codes (cont'd)

Definition. A regular (n, k) LDPC code is a linear block code whose parity-check matrix H contains exactly W_c 1's per column and exactly $W_r = W_c(n/m)$ 1's per row, where $W_c << m$.

Remarks

- note multiplying both sides of $W_c << m$ by n/m implies $W_r << n$.

- the code rate $r = k/n$ can be computed from

$$r = \frac{W_r - W_c}{W_r} \quad = \quad 1 - \frac{W_c}{W_r}$$

$W_c = 3$ is a necessity for good codes (Gallager)

- if H is low density, but the number of 1's per column or row is not constant, the code is an irregular LDPC code

LDPC codes were invented by Robert Gallager of MIT in his PhD dissertation (1960). They received virtually no attention from the coding community until the mid-1990's.

Representation of Linear Block Codes via Tanner Graphs

- one of the very few researchers who studied LDPC codes prior to the recent resurgence is Michael Tanner of UC Santa Cruz

- Tanner considered LDPC codes (and a generalization) and showed how they may be represented effectively by a so-called bipartite graph, now call a Tanner graph

Definition. A bipartite graph is a graph (nodes or vertices connected by undirected edges) whose nodes may be separated into two classes, and where edges may only connect two nodes not residing in the same class

Tanner Graphs (cont'd)

- the two classes of nodes in a Tanner graph are the variable nodes (or bit nodes) and the check nodes (or function nodes)

- the Tanner graph of a code is drawn according to the following rule:

check node j is connected to variable node i
whenever element h_{ji} in H is a 1

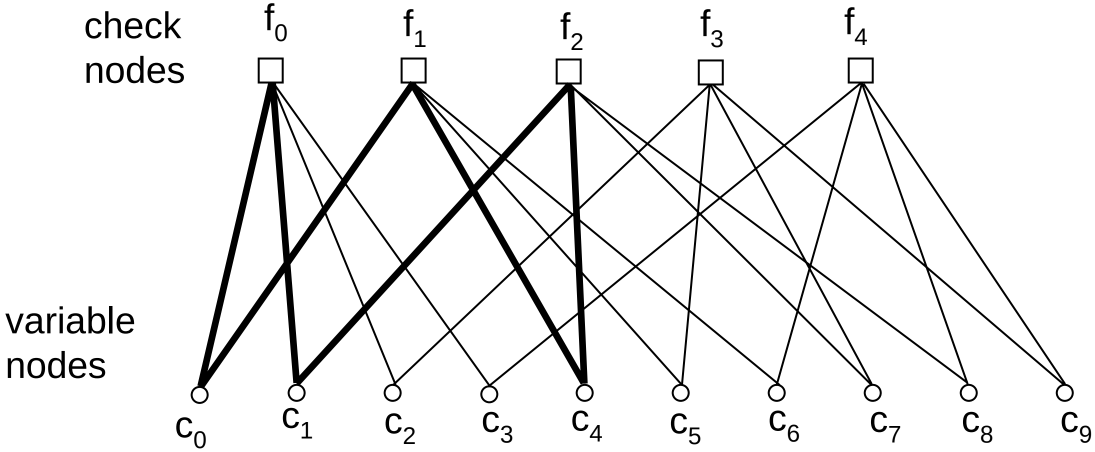
- one may deduce from this that there are $m = n\text{-}k$ check nodes and n variable nodes

- further, the m rows of H specify the m c-node connections, and the n columns of H specify the n v-node connections

Tanner Graphs (cont'd)

Example. (10, 5) block code with $W_c = 2$ and $W_r = W_c(n/m) = 4$.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$



- observe that nodes c_0, c_1, c_2, and c_3 are connected to node f_0 in accordance with the fact that in the first row of H, $h_{00} = h_{01} = h_{02} = h_{03} = 1$ (all others equal zero)

Tanner Graphs (cont'd)

- (for convenience, the first row and first col of *H* are assigned an index of 0)

- observe an analogous situation for $f_{1,}$ $f_{2,}$ $f_{3,}$ and f_4.

- thus, as follows from the fact that $\bar{c}H^T = \bar{0}$, the bit values connected to the same check node must sum to zero

- note that the Tanner graph in this example is regular: each bit node is of degree 2 (has 2 edge connections and each check node is of degree 4)

- this is in accordance with the fact that $W_c = 2$ and $W_r = 4$

- we also see from this why $W_r = W_c(n/m)$ for regular LDPC codes:

(# v-nodes) x (v-node degree) = nW_c
must equal
(#c-nodes) x (c-node degree) = mW_r

Tanner Graphs (cont'd)

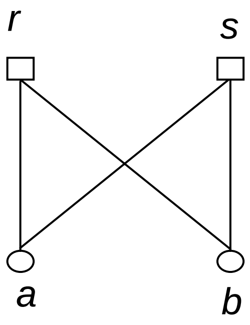
Definition. A cycle of length *l* in a Tanner graph is a path comprising *l* edges which closes back on itself

- the Tanner graph in the above example possesses a length-6 cycle as made evident by the 6 bold edges in the figure

Definition. The girth of a Tanner graph is the minimum cycle length of the graph

- the shortest possible cycle in a bipartite graph is clearly a length-4 cycle

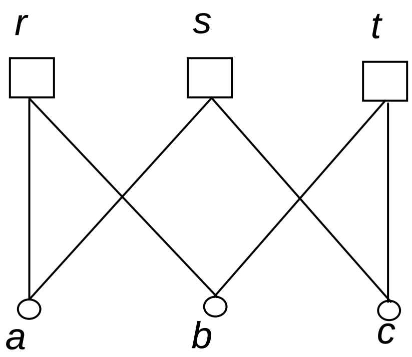
- length-4 cycles manifest themselves in the *H* matrix as four 1's that lie on the corners of a submatrix of *H*:



$$H = \begin{array}{c|ccc|} & a & & b \\ \hline & & \cdots & \\ r & 1 & & 1 \\ & & \cdots & \\ s & 1 & & 1 \\ & & \cdots & \\ \hline \end{array}$$

Tanner Graphs (cont'd)

- length-6 cycles are not quite as easily found in an H matrix:



$$H = \begin{array}{c|ccc} & a & b & c \\ \hline r & 1 & 1 & \\ s & 1 & & 1 \\ t & & 1 & 1 \end{array}$$

- we are interested in cycles, particularly short cycles, because they degrade the iterative decoding algorithm for LDPC codes as will be made evident below

Construction of LDPC Codes

- clearly, the most obvious path to the construction of an LDPC code is via the construction of a low-density parity-check matrix with prescribed properties

- a number of design techniques exist in the literature, and we list a few:

    - Gallager codes (semi-random construction)

    - MacKay codes (semi-random construction)

    - irregular and constrained-irregular LDPC codes (Richardson and Urbanke, Jin and McEliece, Yang and Ryan, Jones and Wesel, ...)

    - finite geometry-based LDPC codes (Kou, Lin, Fossorier)

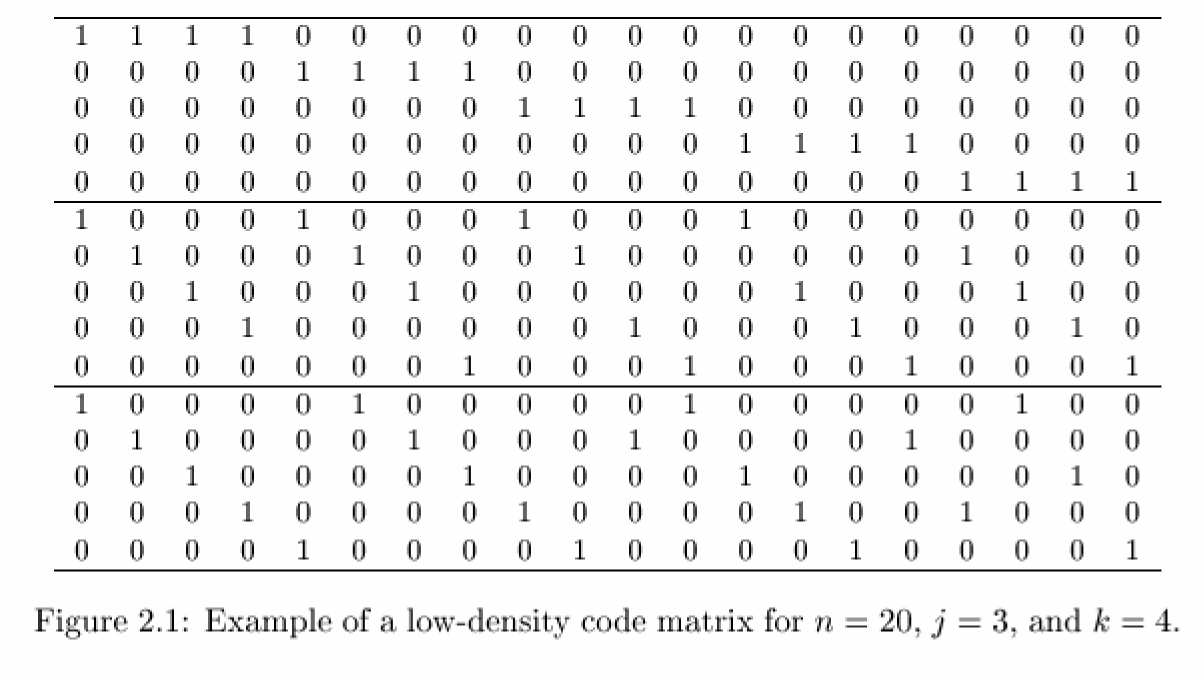
    - combinatorial LDPC codes (Vasic *et al*.)

    - LDPC codes based on array codes (Fan)

Construction of LDPC Codes (cont'd)

Gallager Codes

Example.



1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Figure 2.1: Example of a low-density code matrix for $n = 20$, $j = 3$, and $k = 4$.

captured from Gallager's dissertation

$j = W_c$ and $k = W_r$

Construction of LDPC Codes (cont'd)

Gallager Codes (cont'd)

- The *H* matrix for a Gallager code has the general form:

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \ldots \\ H_{W_c} \end{bmatrix}$$

where H_1 is p x $p \bullet W_r$ and has row weight W_r , and the submatrices H_i are column-permuted versions of H_1.

- Note H has column weight W_c and row weight W_r .

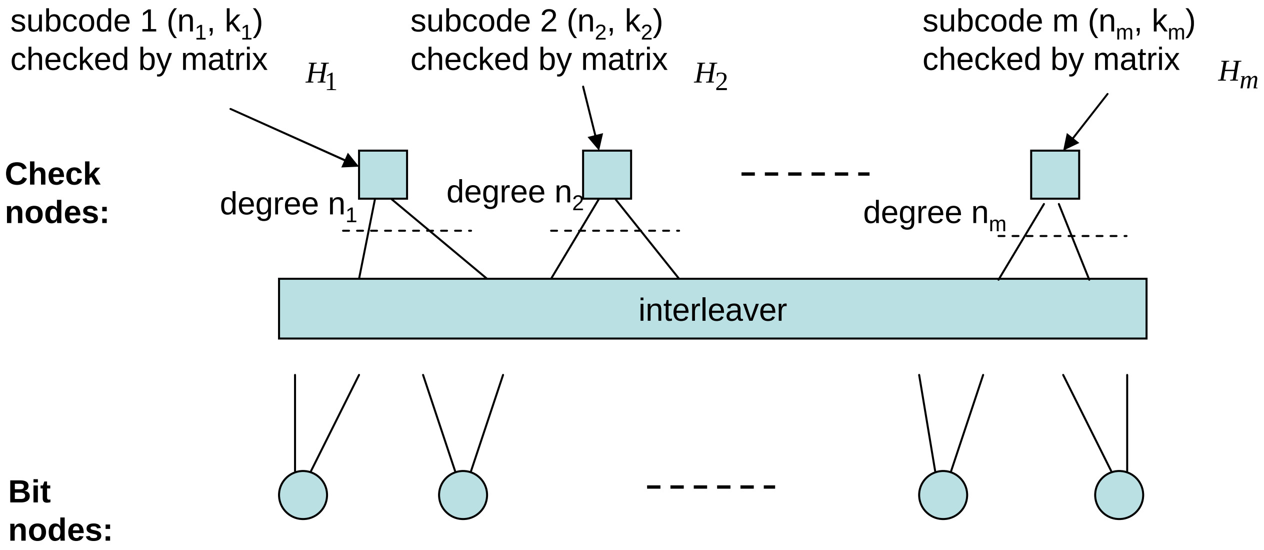
- The permutations must be chosen s. t. length-4 (and higher, if possible) cycles are avoided and the minimum distance of the code is large.

- Codes designs are often performed via computer search. Also see the 2nd edition of Lin and Costello (Prentice-Hall, 2004).

Construction of LDPC Codes (cont'd)

Tanner Codes

- each bit node is associated with a code bit; each "check node" is associated with a subcode whose length is equal to the degree of the node.

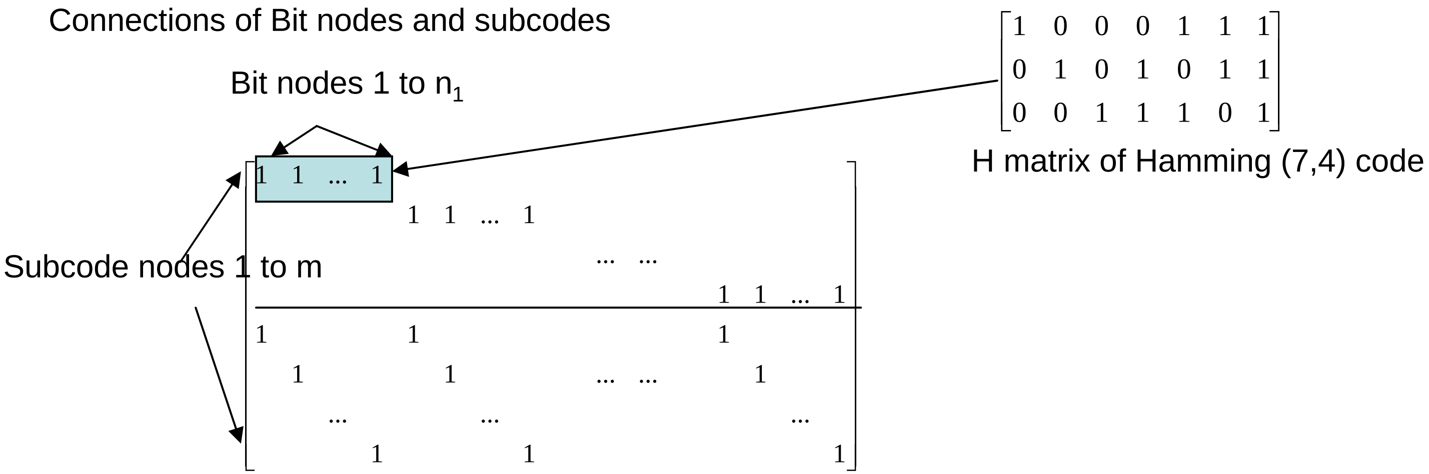


subcode 1 (n_1, k_1) checked by matrix H_1
subcode 2 (n_2, k_2) checked by matrix H_2
subcode m (n_m, k_m) checked by matrix H_m
Check nodes:
degree n_1
degree n_2
degree n_m
interleaver
Bit nodes:

Construction of LDPC Codes (cont'd)

Tanner Codes (cont'd)

Example


Connections of Bit nodes and subcodes
Bit nodes 1 to n_1
Subcode nodes 1 to m
H matrix of Hamming (7,4) code

$$\left[\begin{array}{cccccccccccccc}
1 & 1 & \cdots & 1 & & & & & & & & & & \\
 & & & & 1 & 1 & \cdots & 1 & & & & & & \\
 & & & & & & & & \cdots & \cdots & & & & \\
 & & & & & & & & & & 1 & 1 & \cdots & 1 \\
\hline
1 & & & & 1 & & & & & & 1 & & & \\
 & 1 & & & & 1 & & & \cdots & \cdots & & 1 & & \\
 & & \cdots & & & & \cdots & & & & & & \cdots & \\
 & & & 1 & & & & 1 & & & & & & 1
\end{array}\right]$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Construction of LDPC Codes (cont'd)

MacKay Codes

following MacKay (1999), we list ways to semi-randomly generate sparse matrices H in order of increasing algorithm complexity (but not necessarily improved performance)

1. H generated by starting from an all-zero matrix and randomly inverting W_c not necessarily distinct bits in each column (the resulting LDPC code will be irregular)

2. H generated by randomly creating weight-W_c columns

3. H generated with weight-W_c columns and (as near as possible) uniform row weight

4. H generated with weight-W_c columns, weight-W_r rows, and no two columns having overlap greater than one

5. H generated as in (4), plus short cycles are avoided

6. H generated as in (5), plus $H = [H_1 \; H_2]$ is constrained so that H_2 is invertible (or at least H is full rank)

see http://www.inference.phy.cam.ac.uk/mackay/ for MacKay's large library of codes

Construction of LDPC Codes (cont'd)

MacKay Codes (cont'd)

- frequently an *H* matrix is obtained that is not full rank

- this is generally not a problem which can be seen as follows:

  - once *H* is constructed, we attempt to put it in the form $H = [\, P^{\mathrm{T}} \;\; I \,]_{m\mathrm{x}n}$ via Gauss-Jordan elimination (and possible column swapping) so that we may encode via $G = [\, I \;\; P \,]$

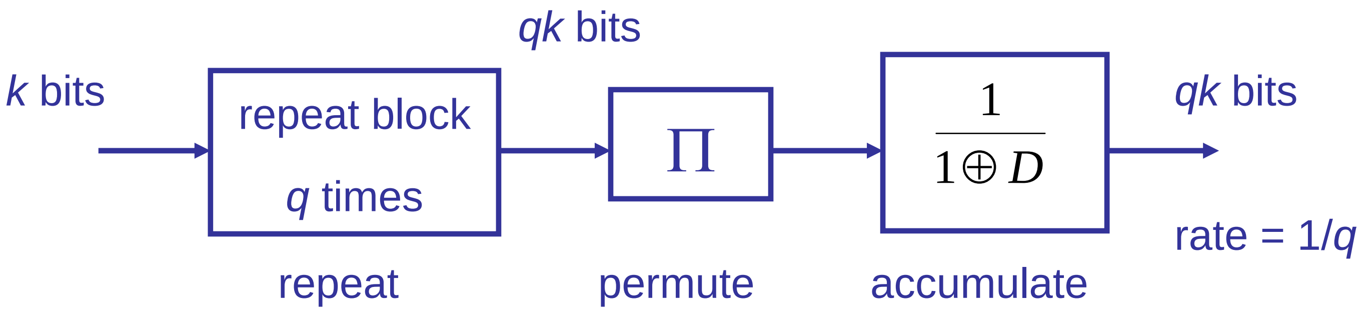
  - if H is not full rank, Gauss-Jordan elimination will result in *H* of the form
$$H = \begin{bmatrix} \tilde{P}^T & I \\ 0 & 0 \end{bmatrix}$$
    where $\tilde{P}^T$ is $m' \,\mathrm{x}\, (n - m')$ and $m' < \mathrm{m}$

- in this latter case, we instead use the matrix $\tilde{H} = [\tilde{P}^T \vdots I]$ which is $m' \,\mathrm{x}\, n$ and corresponds to a higher code rate, $(n - m')/n$, than the original *design rate*, $(n - m)/n$.

Construction of LDPC Codes (cont'd)

Repeat-Accumulate (RA) Codes

- the codes are "turbo-like" and are simple to design and understand (albeit, they are appropriate only for low rates)



- for *q* = 3, $G = \begin{bmatrix} I_k & I_k & I_k \end{bmatrix} \cdot \Pi \cdot A$

where $A = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ & 1 & & 1 \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$ and both *A* and Π are 3*k* x 3*k*.

Construction of LDPC Codes (cont’d)

RA Codes (cont’d)

- for q = 3,

$$H^T = A^{-1} \cdot \Pi^{-1} \cdot \begin{bmatrix} I_k & \\ I_k & I_k \\ & I_k \end{bmatrix}$$

(others are possible)

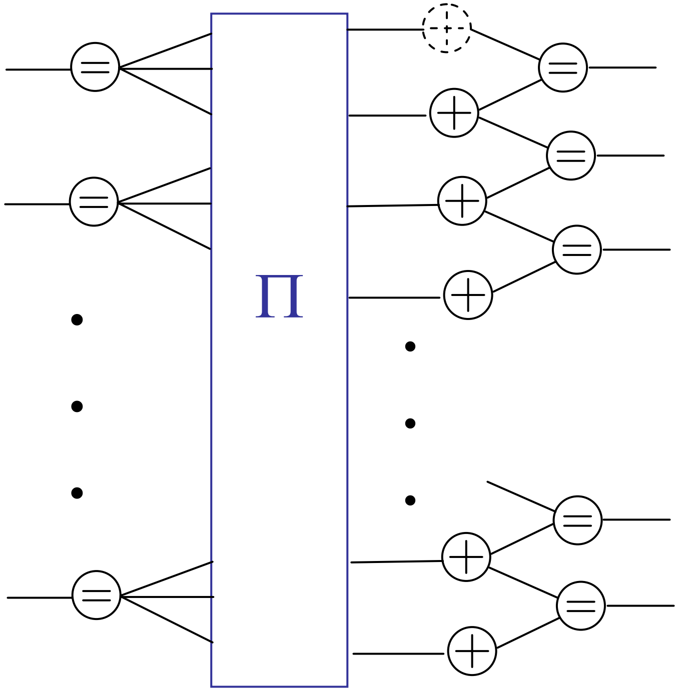
- note since $A \Leftrightarrow \dfrac{1}{1 \oplus D}$, $A^{-1} \Leftrightarrow 1 \oplus D$ so that

$$A^{-1} = \begin{bmatrix} 1 & 1 & & & \\ & 1 & 1 & & \\ & & 1 & \ddots & \\ & & & \ddots & 1 \\ & & & & 1 \end{bmatrix}$$

Construction of LDPC Codes (cont'd)

RA Codes (cont'd)

- an alternative graph to the Tanner graph corresponding to *H* above is



k-bit data word

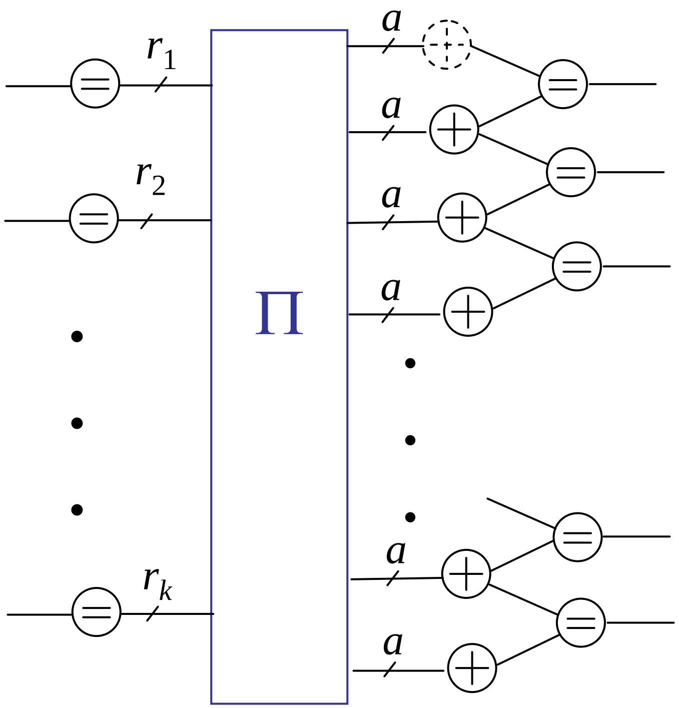
3*k*-bit codeword

Construction of LDPC Codes (cont'd)

Irregular Repeat-Accumulate (IRA) Codes

- graphical representation:

k-bit data word



p-bit parity word

Construction of LDPC Codes (cont'd)

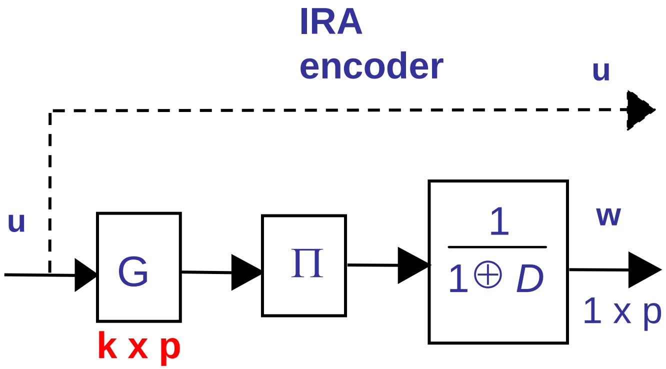
IRA Codes (cont'd)

- non-systematic version:

  codeword is parity word and rate is k/p, $p > k$

- systematic version:

  codeword is data word concatenated with parity word, rate is $k/(k+p)$



Construction of LDPC Codes (cont'd)

Extended Irregular Repeat Accumulate (eIRA) Codes

- *H* is matrix is given below, where the column weight of H_1 is > 2
- note encoding may be performed directly from the *H* matrix by recursively solving for the parity bits

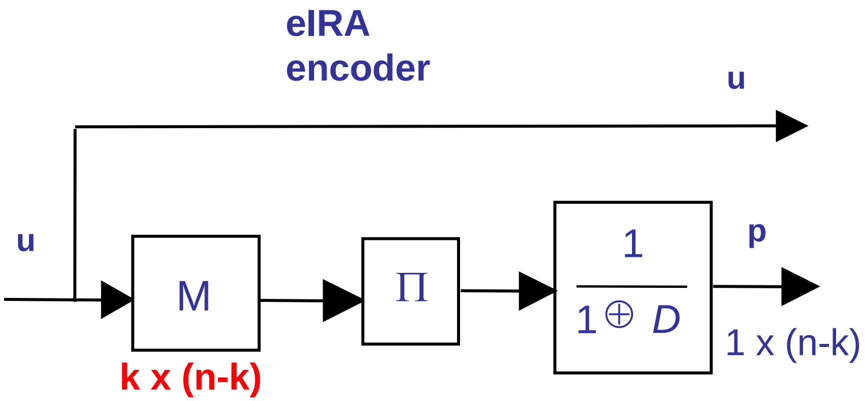
$$H = \left[\begin{array}{cc} H_1 & \underbrace{\begin{matrix} 1 & & & & & \\ 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & 1 & \cdots & & \\ & & & & 1 & \\ & & & & 1 & 1 \end{matrix}}_{A^{-T}} \end{array}\right]$$

- This matrix holds for both eIRA codes and IRA codes. What is different is the size of H_1^T. eIRA: k x $(n-k)$; IRA: k x p, $p > k$ since it is a *G* matrix

Construction of LDPC Codes (cont’d)

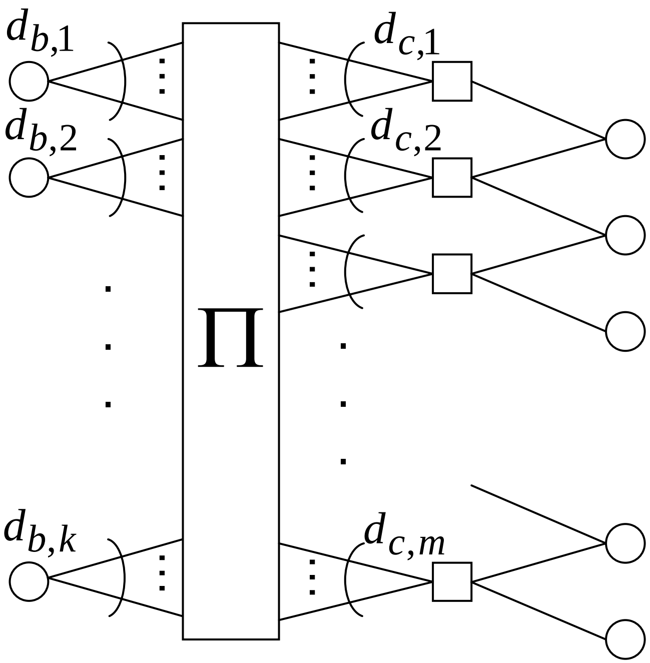
eIRA Codes (cont’d)

- can easily show that $G = [\, I \quad H_1^{\mathrm{T}} A \,]$, from which encoder below follows
- always systematic
- appropriate for high code rates



$$M\,\Pi = H_1^{\mathrm{T}}$$

Accumulator-Based LDPC Codes – Tanner Graph



k info bits

m parity bits

$$m = n - k$$

	$d_{b,i}$	$d_{c,i}$	m, k
RA	q	1	$m > k, m = qk$
IRA	variable	fixed, $\alpha \geq 1$	$m > k$
eIRA	variable	variable	$m \geq 1, k \geq 1$

Construction of LDPC Codes (cont'd)

Array Codes

- The parity-check matrix structure for the class of array codes is

$$H=\begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^{2} & & \alpha^{k-1} \\ I & \alpha^{2} & \alpha^{2\times 2} & & \alpha^{2\times(k-1)} \\ & & & & \\ I & \alpha^{j-1} & \alpha^{(j-1)\times 2} & & \alpha^{(j-1)\times(k-1)} \end{bmatrix} \xrightarrow{\text{simplified}} H=\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & & k-1 \\ 0 & 2 & 4 & & 2(k-1) \\ & & & & \\ 0 & j-1 & (j-1)2 & & (j-1)(k-1) \end{bmatrix}$$

where *I* is the identity matrix and α is a *p*-by-*p* left- (or right-) cyclic shift of the identity matrix *I* by one position (*p* a prime integer), and α^{0} =*I* , *and* $\alpha^{-1}=\text{zero matrix}$.

Example, p = 5:

$$\alpha=\begin{bmatrix} & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \\ 1 & & & & \end{bmatrix}$$

or

$$\alpha=\begin{bmatrix} & & & & 1 \\ 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \end{bmatrix}$$

Construction of LDPC Codes (cont’d)

Array Codes (cont’d)

- Add the *j* x *j* “dual diagonal” submatrix H_p corresponding to the parity check bits
- Delete the first column of the above *H* matrix to remove length-4 cycles
- We obtain the new *H* matrix:

$$H=\begin{bmatrix} 0 & 0 & -1 & \ldots & -1 & 0 & 0 & \ldots & \ldots & 0 \\ -1 & 0 & 0 & & \ldots & 1 & 2 & & & k \\ & & .. & \ldots & & 2 & 4 & & & 2k & . \\ -1 & .. & .. & 0 & 0 & \ldots & \ldots & & & \ldots \\ -1 & \ldots & \ldots & -1 & 0 & j-1 & 2(j-2) & \ldots & .. & (j-1)k \end{bmatrix}$$

- the code rate is *k*/(*j*+*k*), the right *j* x *k* submatrix H_i corresponds to the information bits
- note that we may now encode using the H matrix by solving for the parity bits recursively: **efficient encoding** (more on this below)

Construction of LDPC Codes (cont’d)

Codes Based on Finite Geometries and Combinatorics

- The mathematics surrounding these codes is beyond the scope of this short course.

- Please see:
  - Shu Lin and Daniel Costello, *Error Control Coding*, Prentice-Hall, 2004.
  - The papers by Shu Lin and his colleagues.
  - The papers by Bane Vasic and his colleagues.
  - The papers by Steve Weller and his colleagues.

Construction of LDPC Codes (cont'd)

Codes Designed Using Density Evolution and Related Techniques

- The mathematics surrounding these codes is beyond the scope of this short course.

- Please see:
  - the papers by Richardson and Urbanke (IEEE Trans. Inf. Thy, Feb. 2001)
  - the papers by ten Brink
  - the papers by Wesel

Encoding

- as discussed above, once *H* is generated, it may be put in the form

$$\tilde{H} = [\tilde{P}^T \vdots I]$$

from which the systematic form of the generator matrix is obtained:

$$G = [I \vdots P]$$

- encoding is performed via

$$\bar{c} = \bar{u}G = [\bar{u} \ \vdots \ \bar{u}P],$$

although this is more complex than it appears for capacity-approaching LDPC codes (n large)

Encoding (cont'd)

Example. Consider a (10000, 5000) linear block code. Then $G = [I \vdots P]$ is 5000 x 10000 and *P* is 5000 x 5000. We may assume that the density of ones in *P* is ~ 0.5.

$\Rightarrow$ there are ~ $0.5(5000)^2 = 12.5 \times 10^6$ ones in P

$\Rightarrow$ ~ 12.5×10^6 addition (XOR) operations are required to encode one codeword

Encoding (cont'd)

- Richard and Urbanke (2001) have proposed a lower complexity (linear in the code length) encoding technique based on the *H* matrix (not to be discussed here)

- an alternative approach to simplified encoding is to design the codes via algebraic, geometric, or combinatoric methods

- such "structured" codes are often cyclic or quasi-cyclic and lend themselves to simple encoders based on shift-register circuits

- since they are simultaneously LDPC codes, the same decoding algorithms apply

- often these structured codes lack freedom in the choice of code rate and length

- an alternative to structured LDPC codes are the constrained irregular codes of Jin and McEliece and Yang and Ryan (also called irregular repeat-accumulate (IRA) and extended IRA codes) -- more on this later

Selected Results

- we present here selected performance curves from the literature to demonstrate the efficacy of LDPC codes

- the papers from which these plots were taken are listed in the reference section at the end of the note set

- we indicate the paper each plot is taken from to ensure proper credit is given (references are listed at the end of Part 2).

Selected Results (cont'd)

MacKay (March 1999, Trans IT)

- MacKay (and others) re-invented LDPC codes in the late 1990's
- here are selected figures from his paper (see his paper for code construction details; his codes are regular or nearly regular)

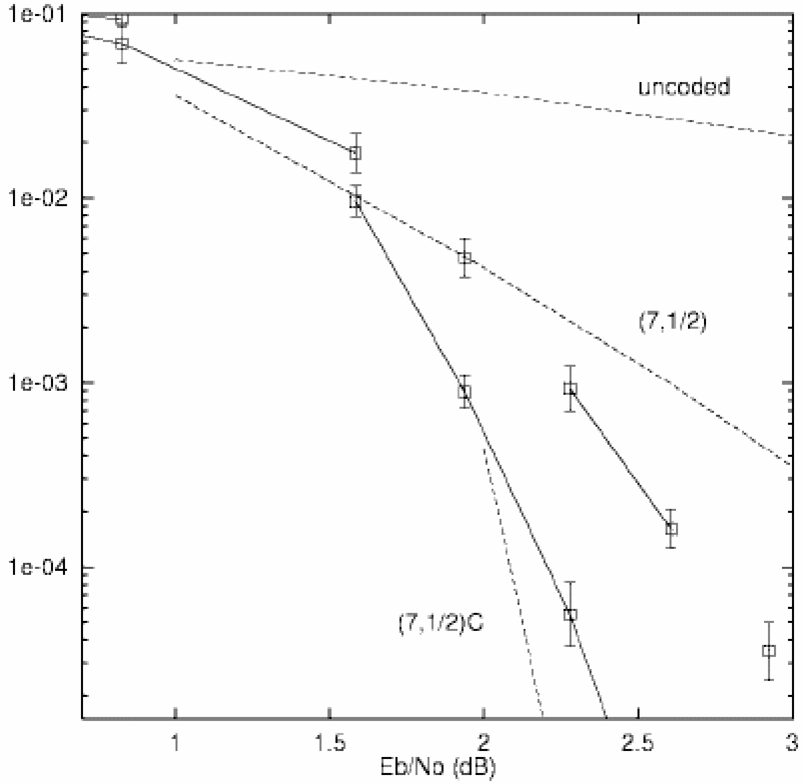
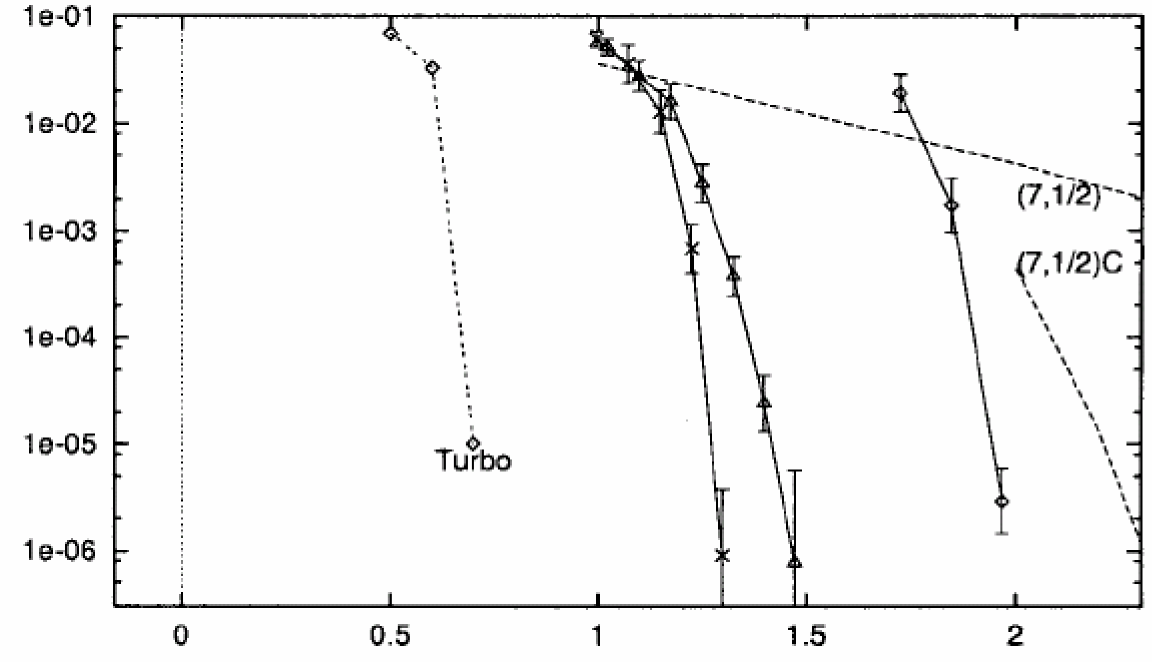


Fig. 11. Short-blocklength Gallager codes' performance over Gaussian channel (solid curves) compared with that of standard textbook codes (dotted curves). Vertical axis shows empirical bit error probability. It should be emphasised that *all* the block errors in the experiments with Gallager codes were *detected* errors: the decoding algorithm reported the fact that it had failed. **Textbook codes:** as in Fig. 9. **Gallager codes:** From left to right the codes had the following parameters (N, K, R): $(1008, 504, 0.5)$ (Construction 1A); $(504, 252, 0.5)$ (1A).

Selected Results (cont'd)

MacKay (cont'd)

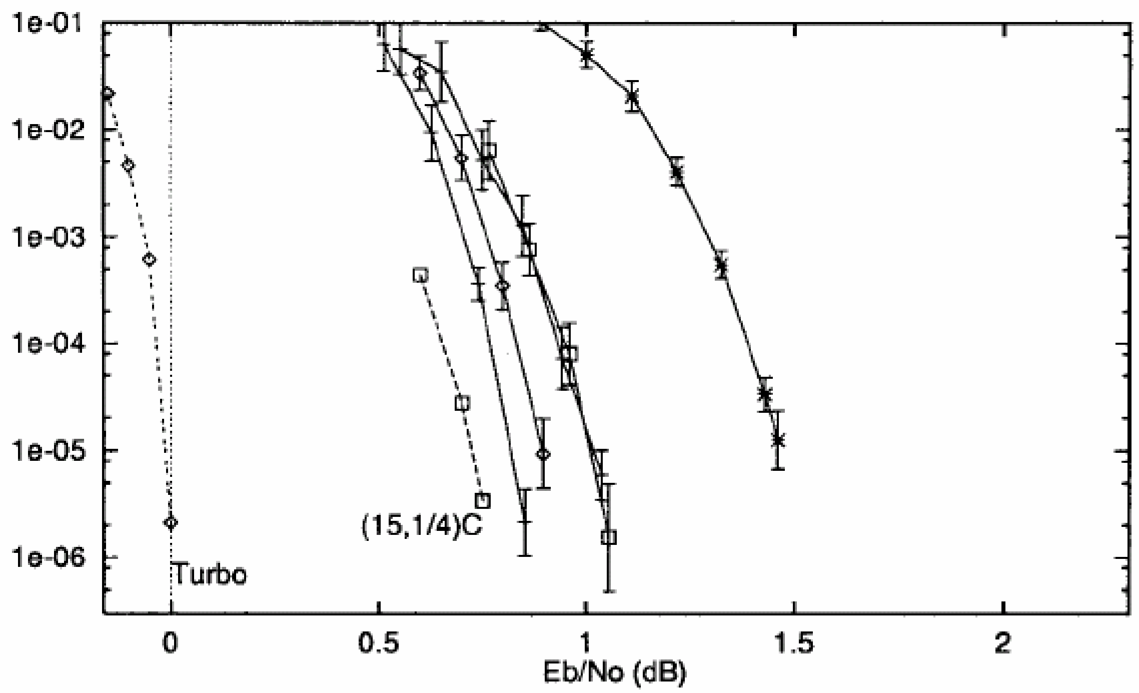


(a)

Fig. 9. Gallager codes' performance over Gaussian channel (solid curves) compared with that of standard textbook codes and state-of-the-art codes (dotted curves). Vertical axis shows empirical bit error probability. It should be emphasised that *all* the block errors in the experiments with Gallager codes were *detected* errors: the decoding algorithm reported the fact that it had failed. Panel (a) shows codes with rates between about $1/2$ and $2/3$; panel (b) shows codes with rates between $1/4$ and $1/3$. **Textbook codes:** The curve labeled $(7, 1/2)$ shows the performance of a rate $1/2$ convolutional code with constraint length 7, known as the *de facto* standard for satellite communications [29]. The curve (7,1/2)C shows the performance of the concatenated code composed of the same convolutional code and a Reed–Solomon code. **State of the art:** The curve (15,1/4)C shows the performance of an extremely expensive and computer intensive concatenated code developed at JPL based on a constraint length 15, rate $1/4$ convolutional code (data courtesy of R. J. McEliece.) The curves labeled Turbo show the performance of the rate $1/2$ Turbo code described in [12], [11] and the rate $1/4$ code reported in [21]. **Gallager codes:** From left to right the codes had the following parameters (N, K, R). Panel (a): $(65389, 32621, 0.499)$ (1B); $(19839, 9839, 0.496)$ (1B); $(29331, 19331, 0.659)$ (1B). Panel (b): $(40000, 10000, 0.25)$ (Construction 2A); $(29507, 9507, 0.322)$ (2B); $(14971, 4971, 0.332)$ (2B); $(15000, 5000, 0.333)$ (2A); $(13298, 3296, 0.248)$ (1B).

Selected Results (cont’d)

MacKay (cont’d)



(b)

Fig. 9. Gallager codes’ performance over Gaussian channel (solid curves) compared with that of standard textbook codes and state-of-the-art codes (dotted curves). Vertical axis shows empirical bit error probability. It should be emphasised that *all* the block errors in the experiments with Gallager codes were *detected* errors: the decoding algorithm reported the fact that it had failed. Panel (a) shows codes with rates between about $1/2$ and $2/3$; panel (b) shows codes with rates between $1/4$ and $1/3$. **Textbook codes:** The curve labeled $(7, 1/2)$ shows the performance of a rate $1/2$ convolutional code with constraint length 7, known as the *de facto* standard for satellite communications [29]. The curve (7,1/2)C shows the performance of the concatenated code composed of the same convolutional code and a Reed–Solomon code. **State of the art:** The curve (15,1/4)C shows the performance of an extremely expensive and computer intensive concatenated code developed at JPL based on a constraint length 15, rate $1/4$ convolutional code (data courtesy of R. J. McEliece.) The curves labeled Turbo show the performance of the rate $1/2$ Turbo code described in [12], [11] and the rate $1/4$ code reported in [21]. **Gallager codes:** From left to right the codes had the following parameters (N, K, R). Panel (a): $(65389, 32621, 0.499)$ (1B); $(19839, 9839, 0.496)$ (1B); $(29331, 19331, 0.659)$ (1B). Panel (b): $(40000, 10000, 0.25)$ (Construction 2A); $(29507, 9507, 0.322)$ (2B); $(14971, 4971, 0.332)$ (2B); $(15000, 5000, 0.333)$ (2A); $(13298, 3296, 0.248)$ (1B).

Selected Results (cont'd)

Irregular LDPC Codes

- our discussions above favored regular LDPC codes for their simplicity, although we gave examples of irregular LDPC codes

- recall an LDPC code is irregular if the number of 1's per column of *H* and/or the number of 1's per row of *H* varies

- in terms of the Tanner graph, this means that the v-node degree and/or the c-node degree is allowed to vary (the degree of a node is the number of edges connected to it)

- a number of researchers have examined the optimal degree distribution among nodes:

  - MacKay, Trans. Comm., October 1999
  - Luby, et al., Trans. IT, February 2001
  - Richardson, et al., Trans. IT, February 2001
  - Chung, et al., Comm. Letters, February 2001

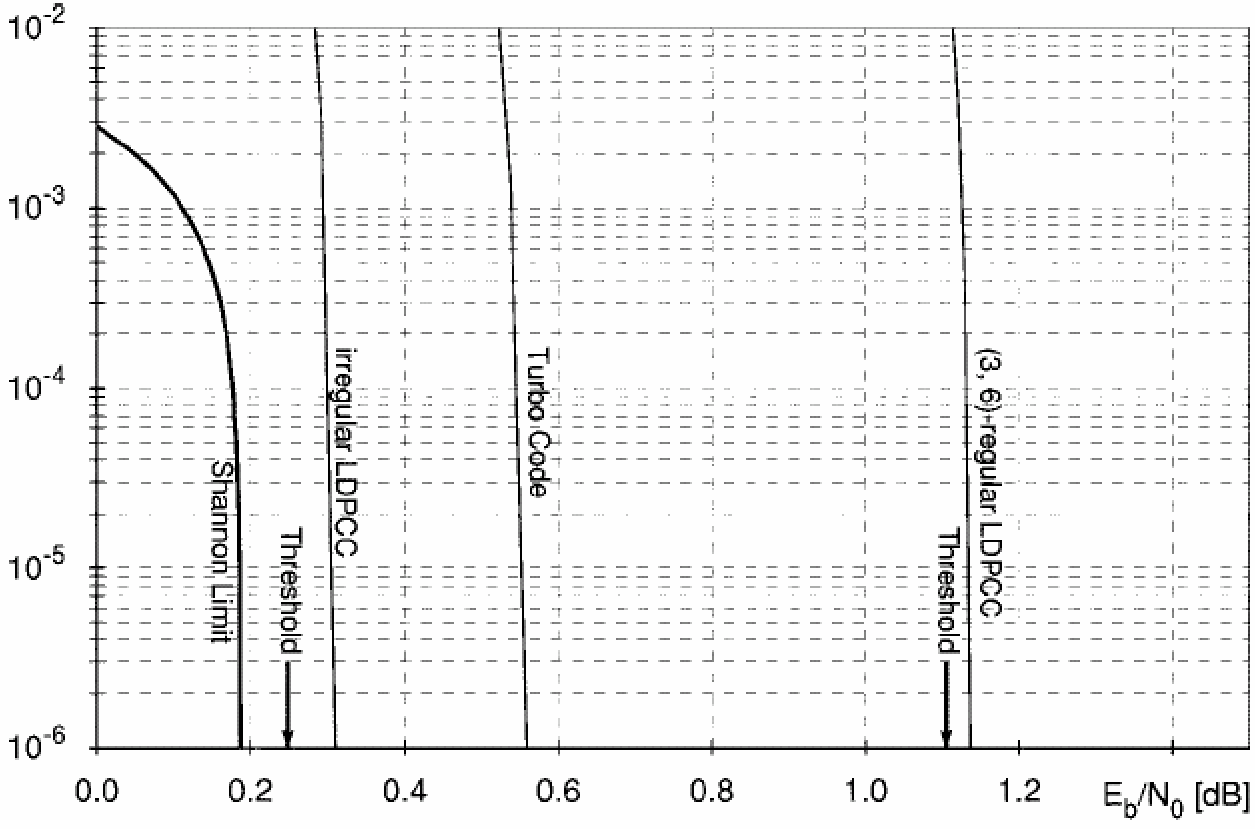
- the results have been spectacular, with performance surpassing the best turbo codes

Selected Results (cont'd)

Richardson et al. Irregular Codes

• the plots below are for a (3, 6) – regular LDPC code, an optimized irregular LDPC code, and a turbo code

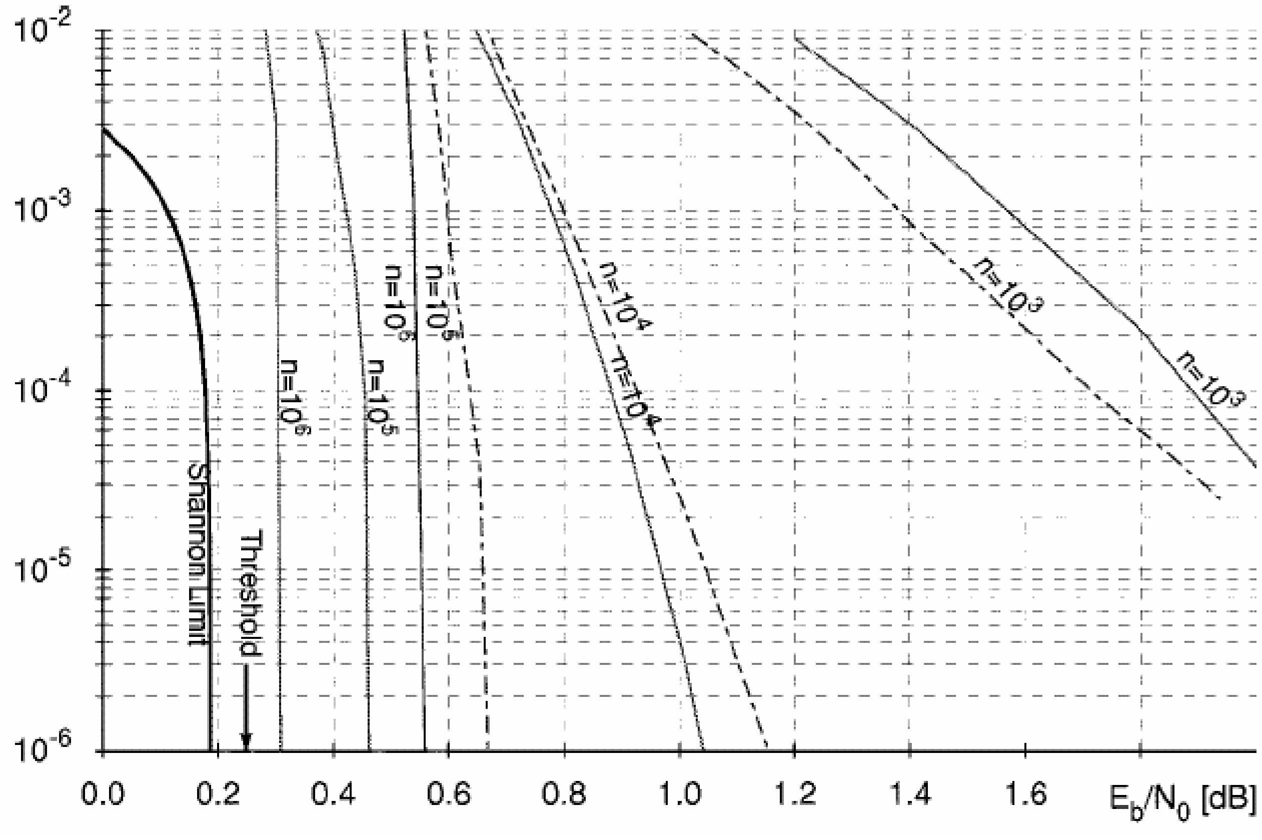
• the code parameters are ½ (10^6, $10^6/2$) in all cases



Selected Results (cont’d)

Richardson et al. Irregular Codes (cont’d)

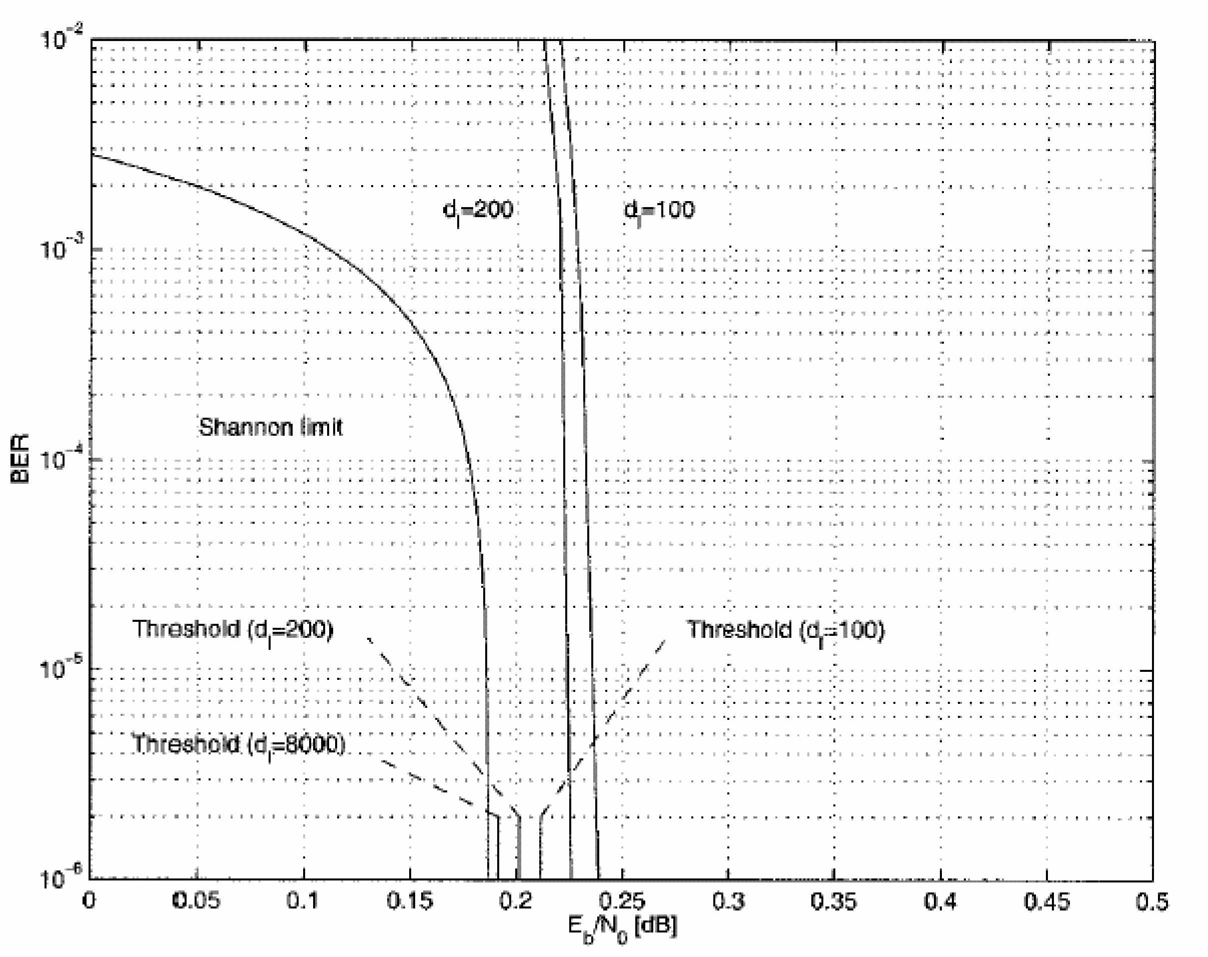
- plot below: turbo codes (dashed) and irregular LDPC codes (solid); for block lengths of n=10^3, 10^4, 10^5, and 10^6; all rates are ½



Selected Results (cont'd)

Chung et al. Irregular LDPC Code

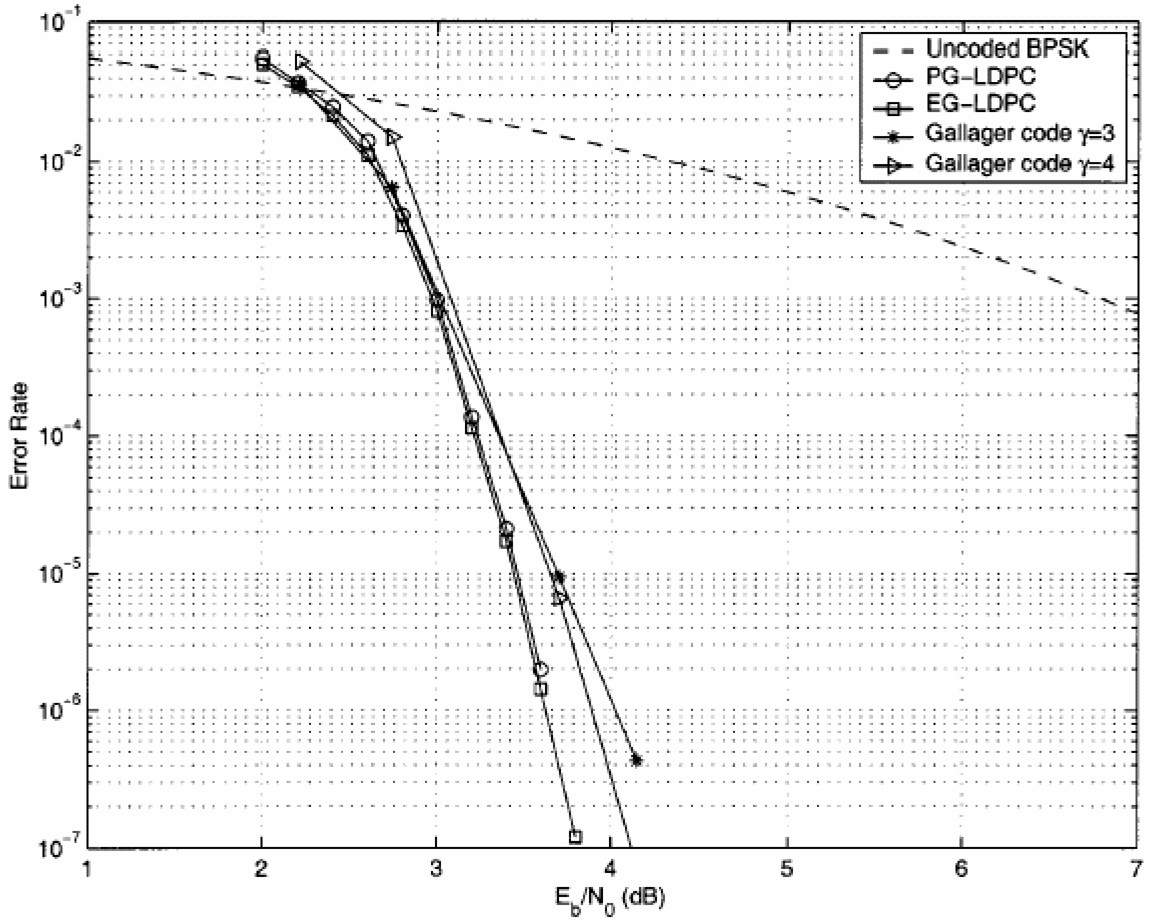
- the plot below is of two separate ½ (10^7, $10^7/2$) irregular LDPC codes



Selected Results (cont'd)

Kou et al. LDPC Codes (IEEE Trans. IT, Nov 2001)

- LDPC code based on Euclidean geometries (EG): (1023, 781)
- LDPC code based on Projective geometries (PG):
- Gallager code: (1057, 813)



Selected Results (cont'd)

Kou et al. (cont'd)

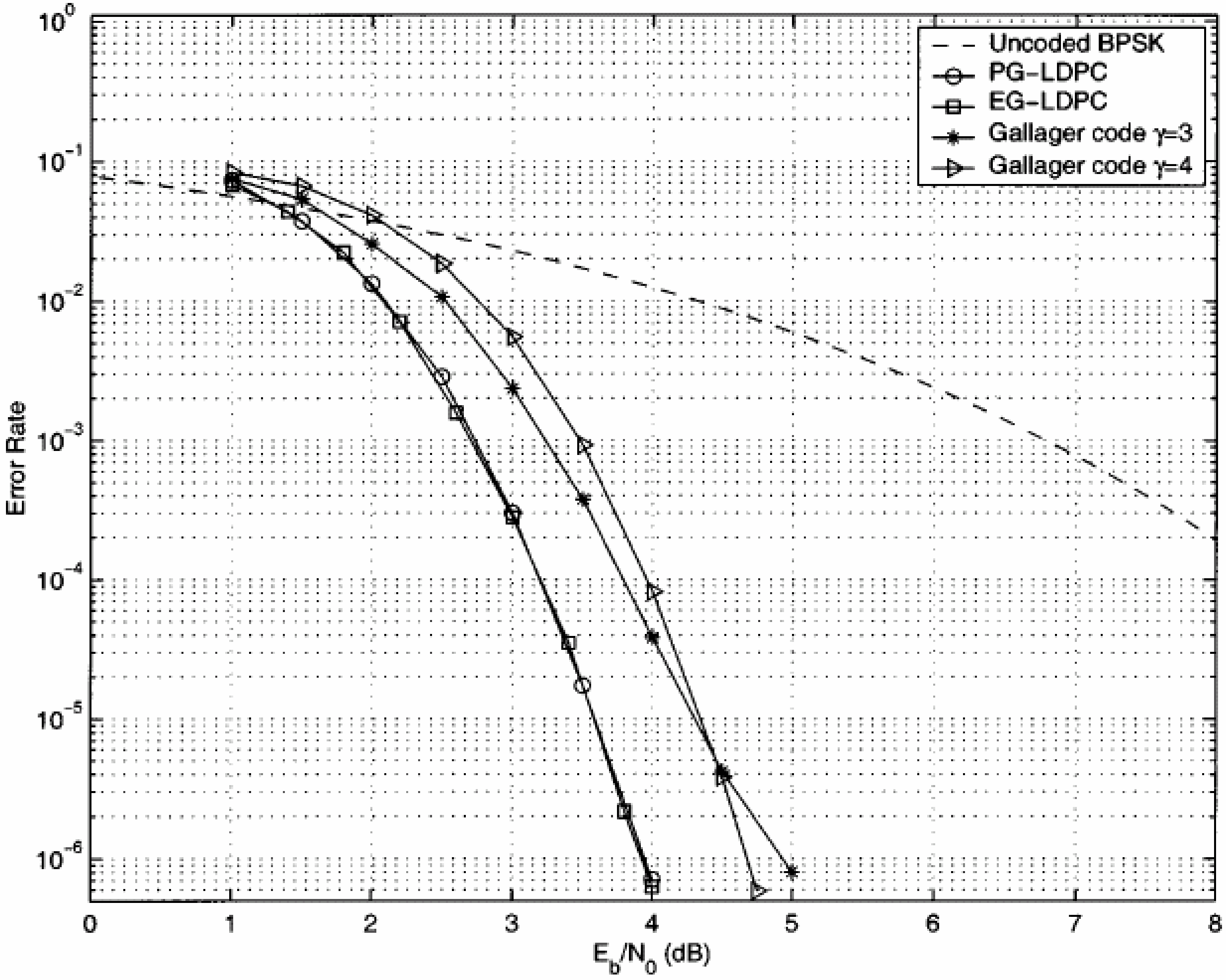


Fig. 2. Bit-error probabilities of the (255, 175) EG-LDPC code, (273, 191) PG-LDPC code and two computer generated (273, 191) Gallager codes with the SPA decoding.

Selected Results (cont'd)

Kou et al. (cont'd)

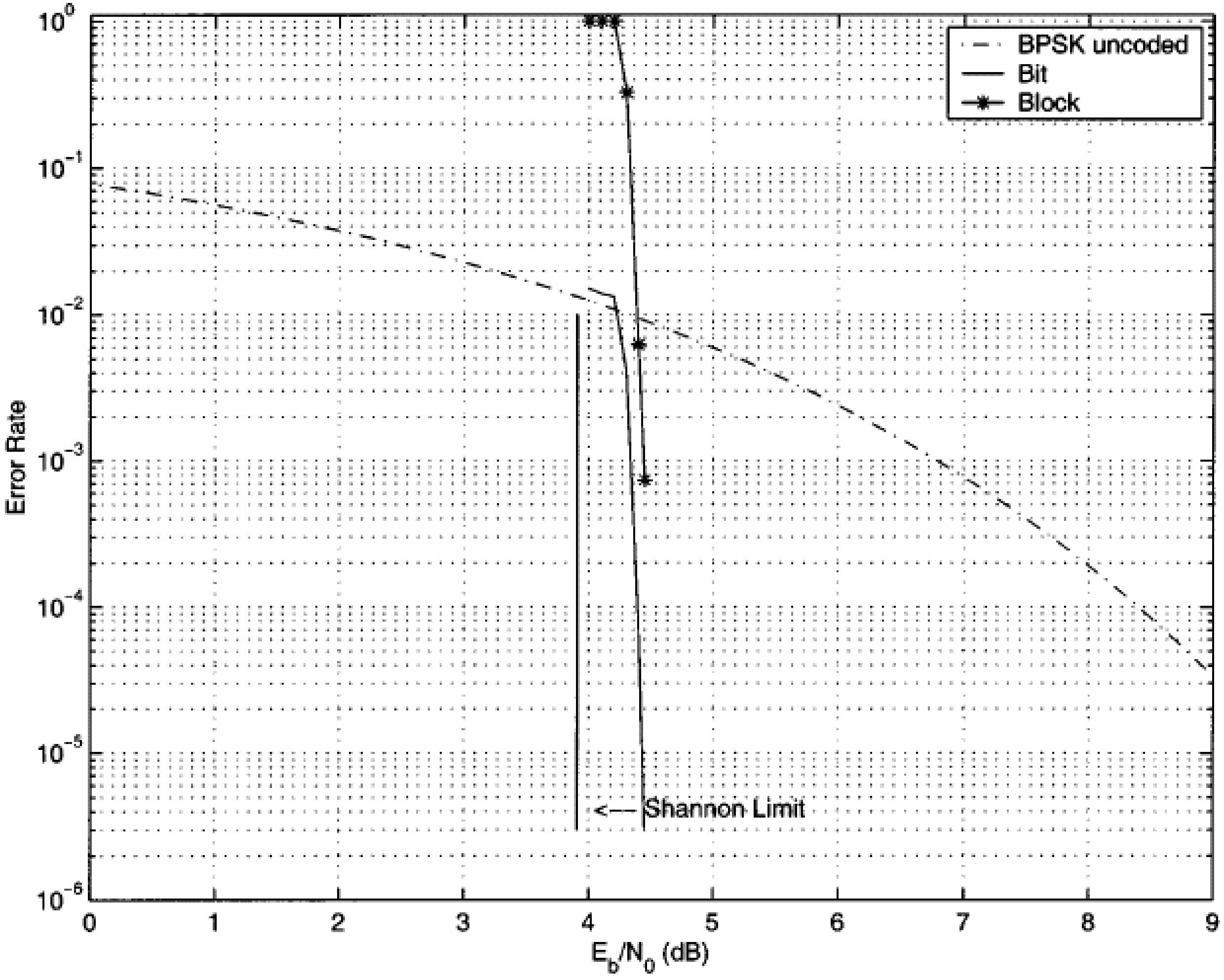
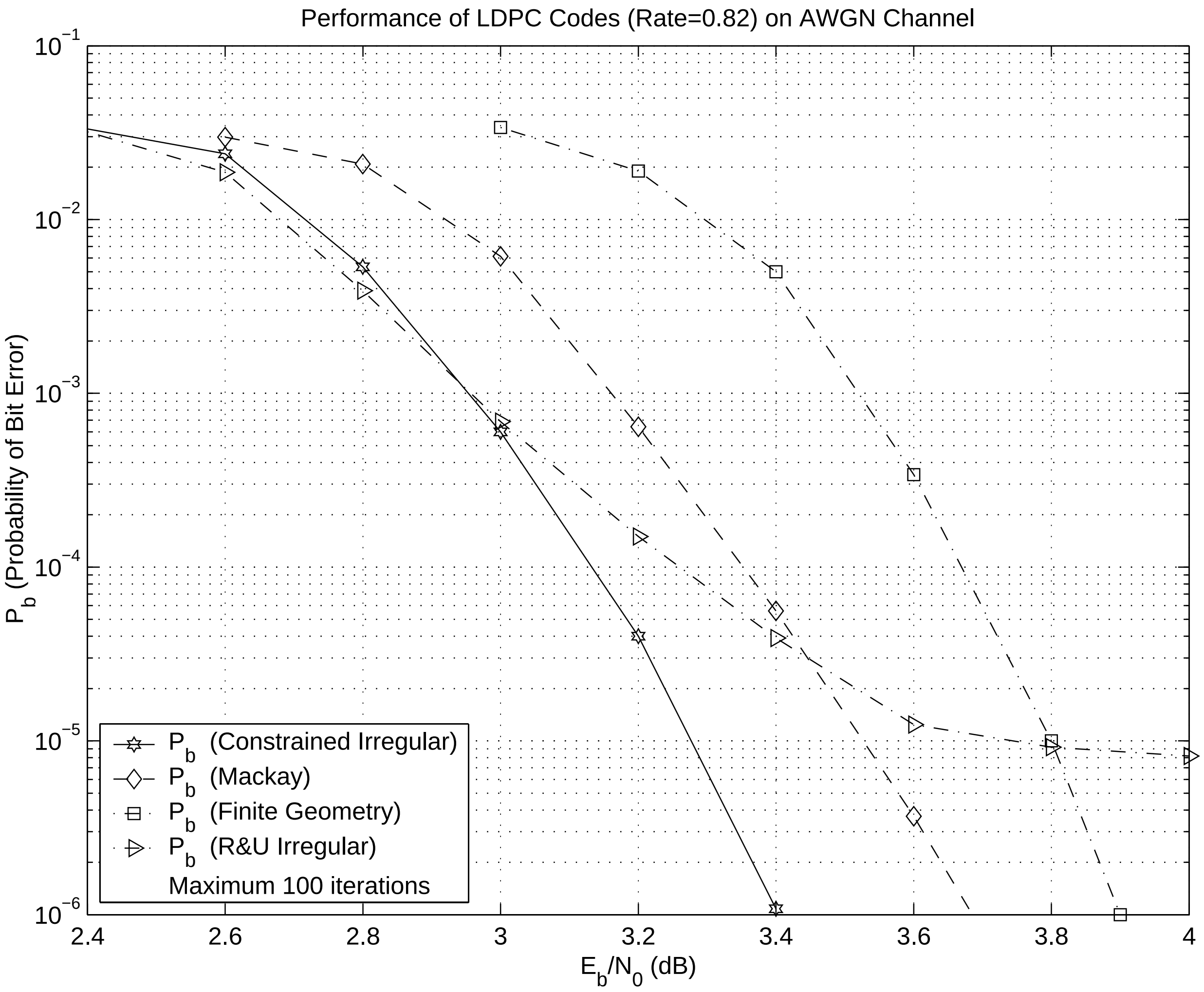


Fig. 10. Bit- and block-error probabilities of the extended (65520, 61425) EG-LDPC code with the SPA decoding.

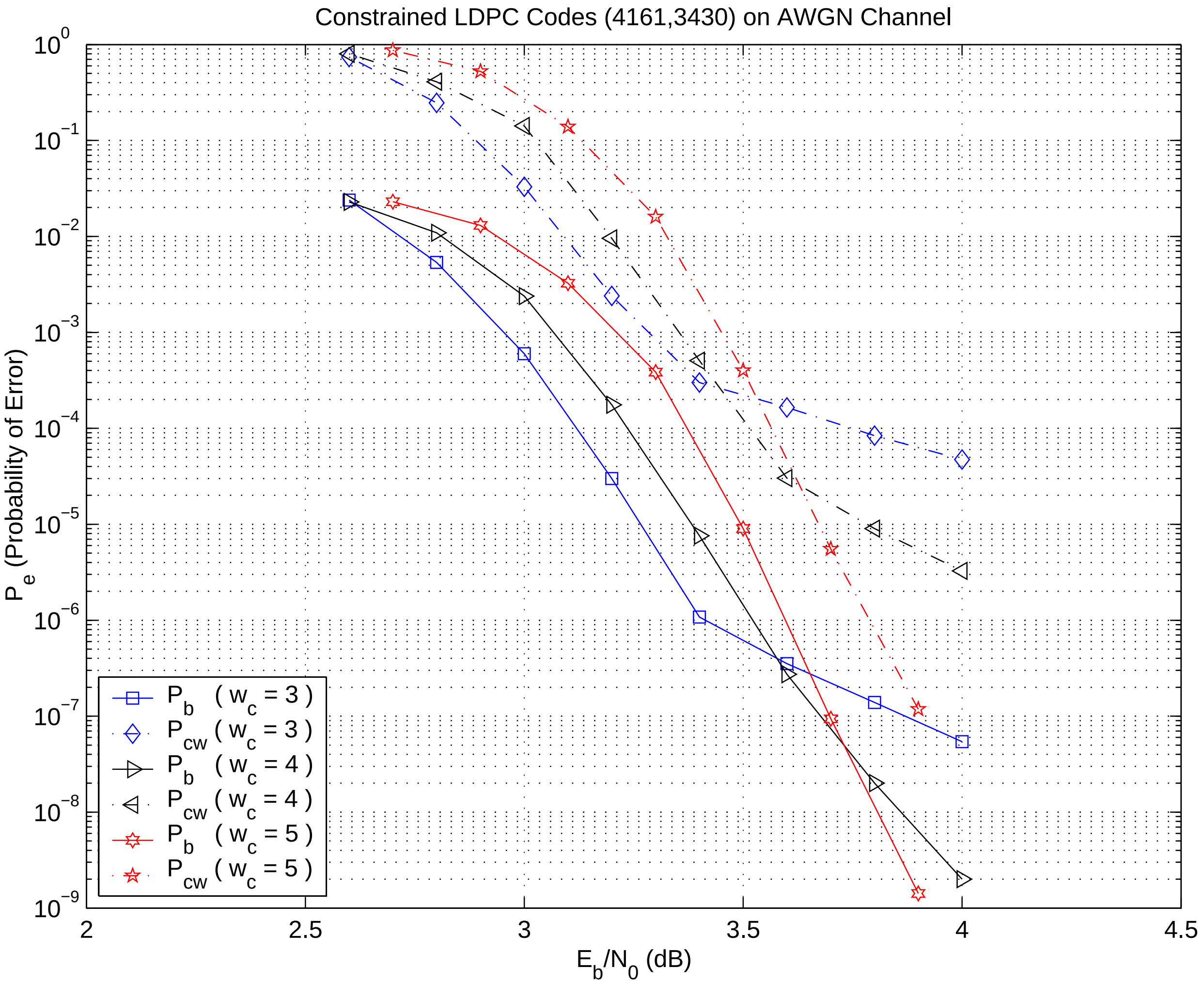
Selected Results (cont'd)

Extended IRA Code Results (Yang, Ryan, and Li - 2004)



Selected Results (cont’d)

Extended IRA Code Results (cont’d)



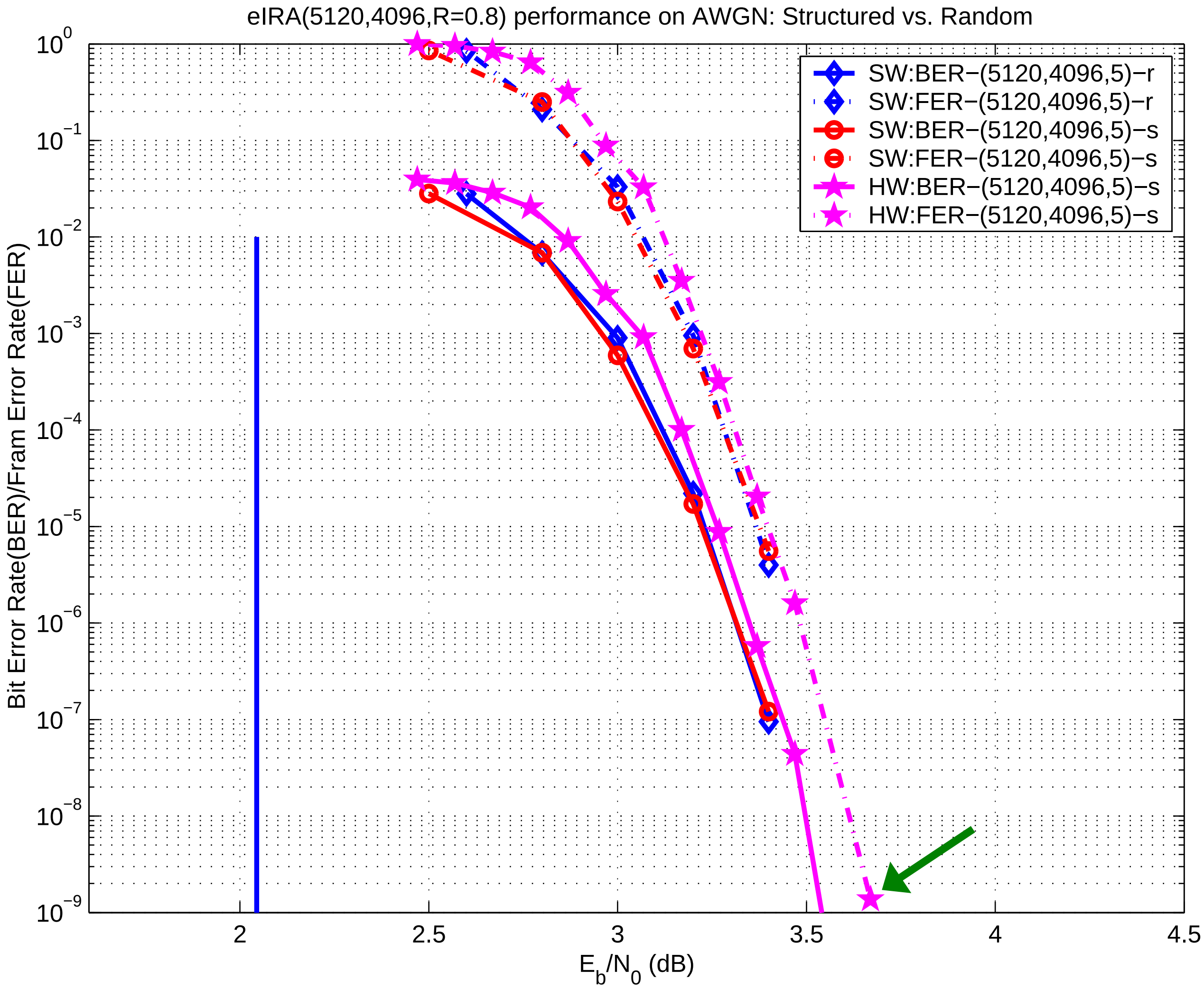
SeIRA Code - Example 3

Rate-0.8 (5120, 4096) SeIRA code

Rate-0.8 (5120, 4096) unstructured eIRA code

HW simulation results for the 0.8(5120, 4096) structured code.

The last data point for HW simulation corresponds to 1 error event in 730 million simulated codewords

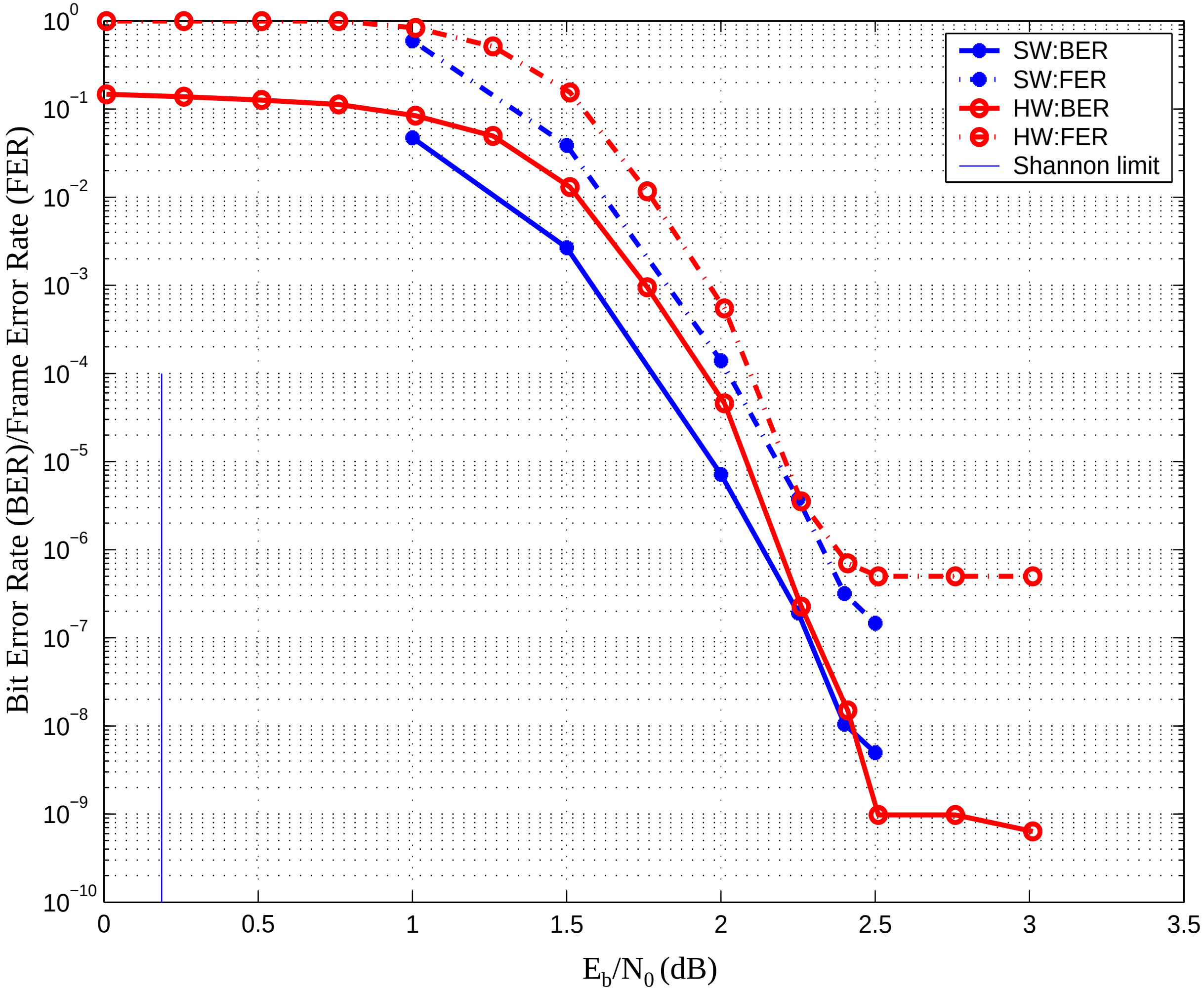


SeIRA Code - Example 4

Rate-0.5, (2048, 1024) SeIRA code

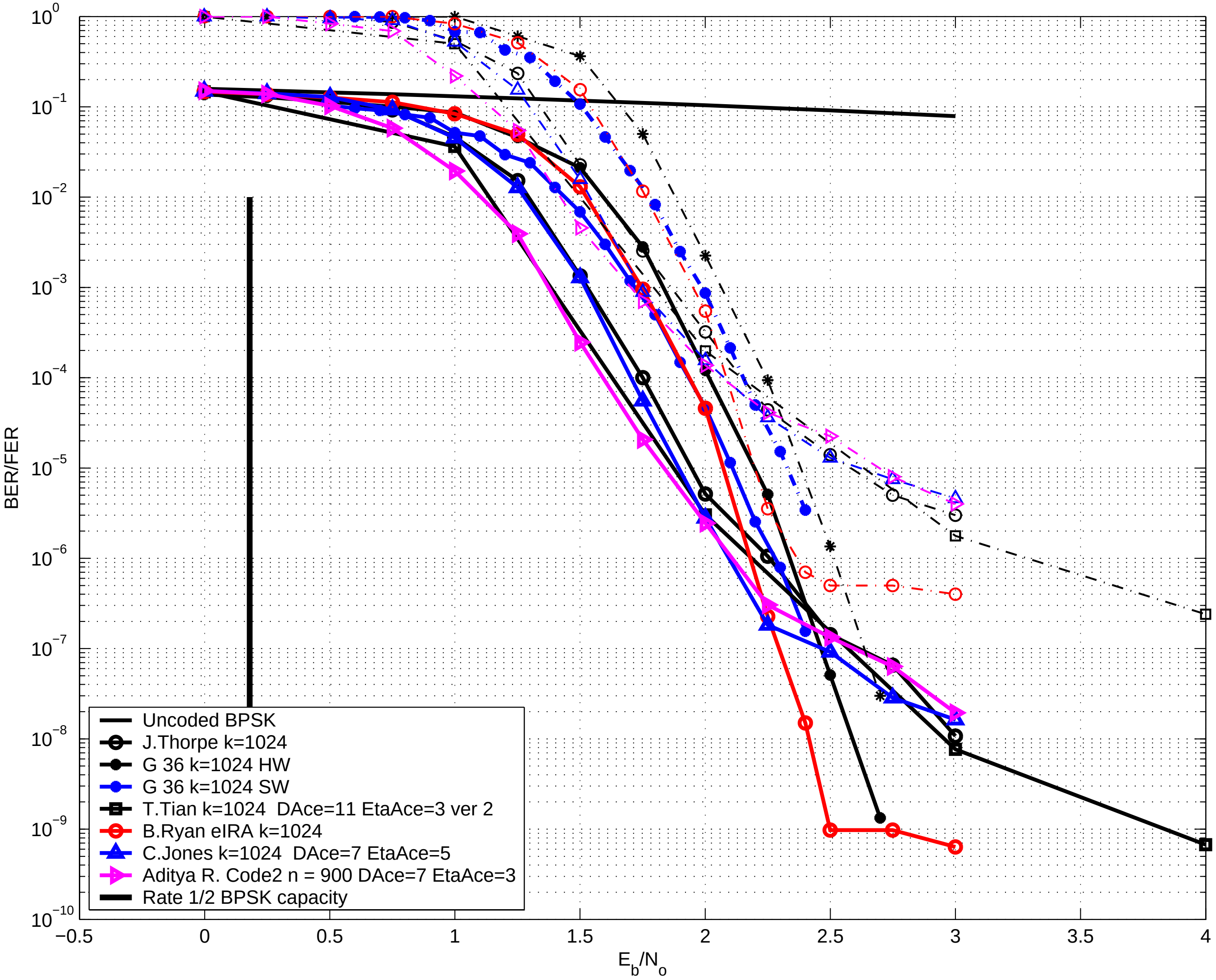
Hardware (HW) and software (SW) simulation results

Floor at 1e-9 due to trapping sets (not d_{min})



SeIRA Code - Example 4 (cont'd)

Hardware simulations of 0.5(2048, 1024) LDPC codes



Acknowledgment:

Chris Jones, JPL

Low-Density Parity-Check Codes

Part II - The Iterative Decoder

William Ryan, Professor

Electrical and Computer Engineering Department
The University of Arizona

Box 210104
Tucson, AZ 85721

ryan@ece.arizona.edu

2005

Decoding Overview

- in addition to presenting LDPC codes in his seminal work in 1960, Gallager also provided a decoding algorithm that is effectively optimal

- since that time, other researchers have independently discovered that algorithm and related algorithms, albeit sometimes for different applications

- the algorithm iteratively computes the distributions of variables in graph-based models and comes under different names, depending on the context:

  - sum-product algorithm
  - min-sum algorithm (approximation)
  - forward-backward algorithm, BCJR algorithm (trellis-based graphical models)
  - belief-propagation algorithm, message-passing algorithm (machine learning, AI, Bayesian networks)

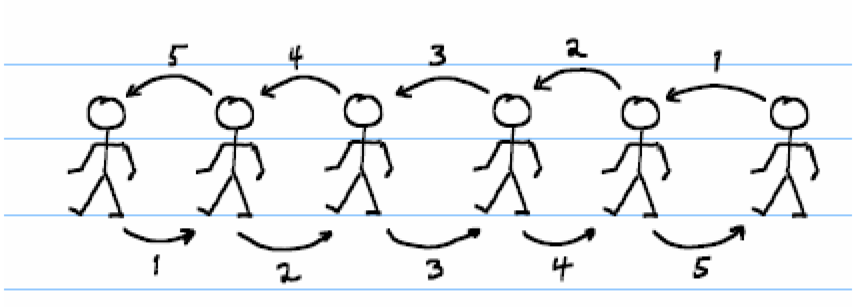
- the iterative decoding algorithm for turbo codes has been shown by McEliece (1998) and others to be a specific instance of the sum-product/belief-propagation algorithm

  - the “sum-product," "belief propagation," and "message passing" all seem to be commonly used for the algorithm applied to the decoding of LDPC codes

Example: Distributed Soldier Counting {Pearl, 1988)

A. Soldiers in a line.

- Counting rule: Each soldier receives a number from his right (left), adds one for himself, and passes the sum to his left (right).
- Total number of soldiers = (incoming number) + (outgoing number)



(one side)

B. Soldiers in a Y Formation

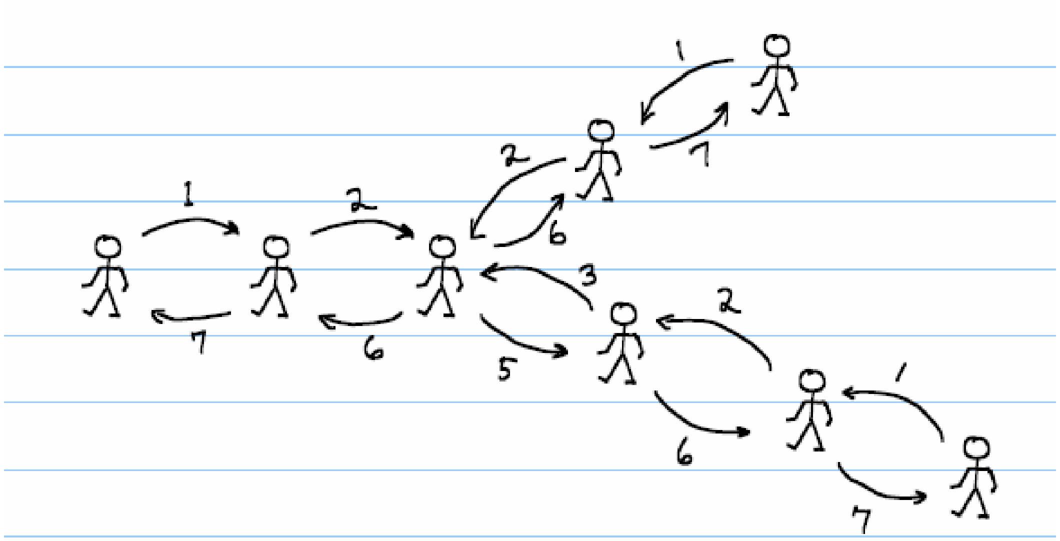
- Counting rule: The “message” that soldier X passes to soldier Y is the sum of all incoming messages, plus one for soldier X, minus soldier Y’s message

$$I_{X\to Y} = \sum_{Z\in n(X)} I_{Z\to X} - I_{Y\to X} + I_X$$

$$= \sum_{Z\in n(X)\backslash Y} I_{Z\to X} + I_X$$

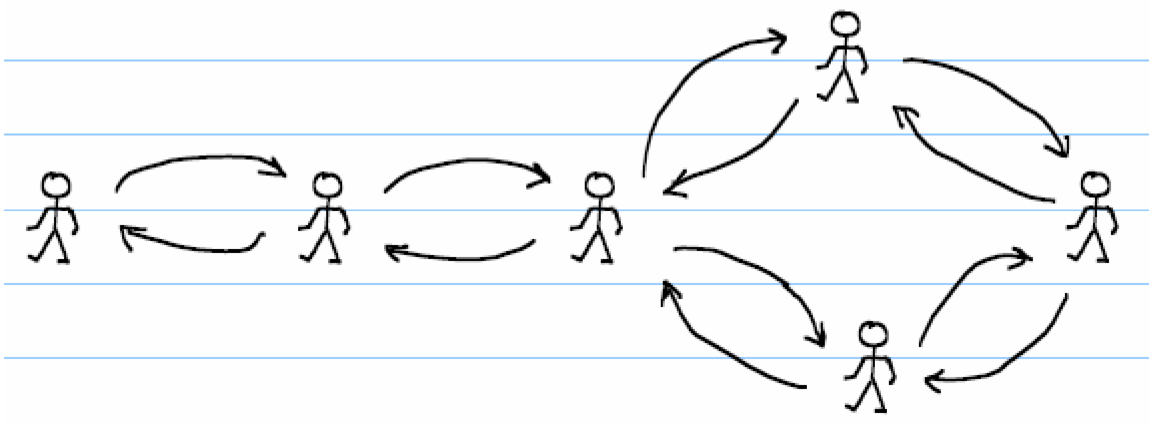
$$= \text{extrinsic information} + \text{intrinsic information}$$

- Total number of soldiers = (message soldier X passes to soldier Y) + (message soldier Y passes to soldier X)

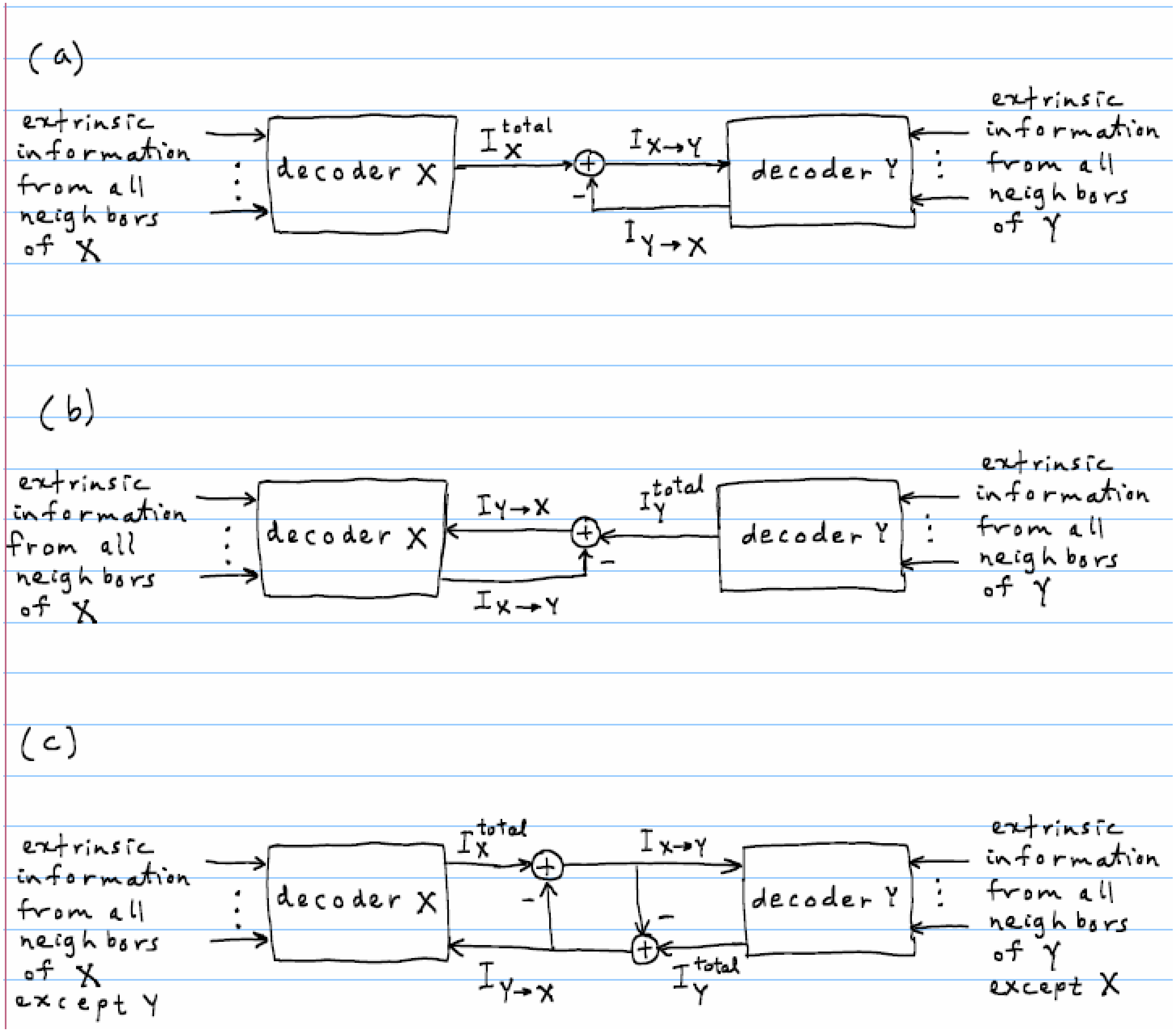


C. Formation Contains a Cycle

- The situation is untenable: No viable counting strategy exists; there is also a positive feedback effect within the cycle and the count tends to infinity.
- Conclusion: message-passing decoding cannot be optimal when the code's graph contains a cycle



The Turbo Principle


(a)
extrinsic information from all neighbors of X
decoder X
I_X^{total}
$I_{X\to Y}$
−
decoder Y
$I_{Y\to X}$
extrinsic information from all neighbors of Y
(b)
extrinsic information from all neighbors of X
decoder X
$I_{Y\to X}$
I_Y^{total}
−
decoder Y
$I_{X\to Y}$
extrinsic information from all neighbors of Y
(c)
extrinsic information from all neighbors of X except Y
decoder X
I_X^{total}
$I_{X\to Y}$
−
−
decoder Y
$I_{Y\to X}$
I_Y^{total}
extrinsic information from all neighbors of Y except X

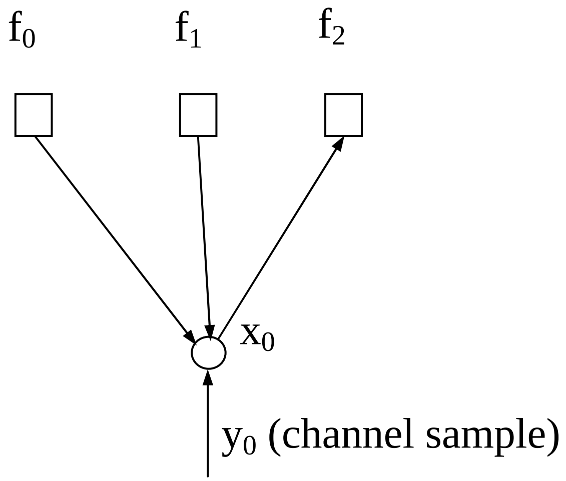
The Turbo Principle Applied to LDPC Decoding

- the concept of extrinsic information is helpful in the understanding of the sum-product/message-passing algorithm (the messages to be passed are extrinsic information)

- we envision Tanner graph edges as information-flow pathways to be followed in the iterative computation of various probabilistic quantities

- this is similar to (a generalization of) the use of trellis branches as paths in the Viterbi algorithm implementation of maximum-likelihood sequence detection/decoding

- Consider the subgraph



the information passed concerns

- $\Pr(x_0 = +1 \mid \mathbf{y})$ or
- $\Pr(x_0 = +1 \mid \mathbf{y}) / \Pr(x_0 = -1 \mid \mathbf{y})$ or
- $\log[\Pr(x_0 = +1 \mid \mathbf{y}) / \Pr(x_0 = -1 \mid \mathbf{y})]$

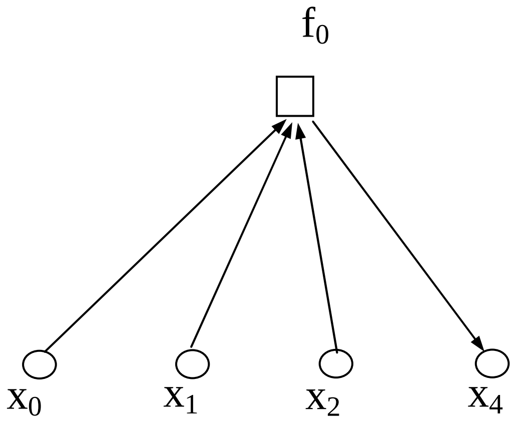
- The arrows indicate the situation for $x_0 \rightarrow f_2$

- All of the information that x_0 possesses is sent to node f_2, except for the information that node f_2 already possesses (extrinsic information)

- In one half-iteration of the decoding algorithm, such computations, $x_i \rightarrow f_j$, are made for all v-node/c-node pairs.

- In the other half-iteration, messages are passed in the opposite direction: from c-nodes to v-nodes, $f_j \rightarrow x_i$.

- Consider the subgraph corresponding to the other half-iteration



the information passed concerns

Pr(check equation f_0 is satisfied)

- The arrows indicate the situation for $f_0 \rightarrow x_4$

- node f_0 passes all (extrinsic) information it has available to it to each of the bit nodes x_i, excluding the information the receiving node already possesses.

- only information consistent with $c_0 + c_1 + c_2 + c_4 = 0$ is sent

Probability-Domain Decoder

- much like optimal (MAP) symbol-by-symbol decoding of trellis codes, we are interested in computing *the a posteriori probability* (APP) that a given bit in $\bar{c}$ equals one, given the received word $\bar{y}$ and the fact that $\bar{c}$ must satisfy some constraints

- without loss of generality, let us focus on the decoding of bit *c_i* ; thus we are interested in computing

$$\Pr\left(c_i = 1 \,\middle|\, \bar{y}, S_i\right)$$

where S_i is the event that the bits in $\bar{c}$ satisfy the W_c parity-check equations involving c_i

- later we will extend this to the more numerically stable computation of the log-APP ratio, also call the log-likelihood ratio (LLR):

$$\log\left(\frac{\Pr(c_i = 0 | \bar{y}, S_i)}{\Pr(c_i = 1 | \bar{y}, S_i)}\right)$$

Lemma 1 (Gallager)

- consider a sequence of *m* independent binary digits $\bar{a} = (a_1, \ldots, a_m)$ in which *Pr(a*$_k$*=1) = p*$_k$.

- Then the probability that $\bar{a}$ contains an *even* number of 1's is

$$\frac{1}{2} + \frac{1}{2}\prod_{k=1}^{m}(1 - 2p_k) \qquad (*)$$

- The probability that $\bar{a}$ contains an *odd* number of 1's is one minus this value:

$$\frac{1}{2} - \frac{1}{2}\prod_{k=1}^{m}(1 - 2p_k)$$

proof: Induction on *m*.

Notation

- V_j = {v-nodes connected to c-node j}
- $V_j \setminus i$ ={v-nodes connected to c-node j} \ {v-node i}

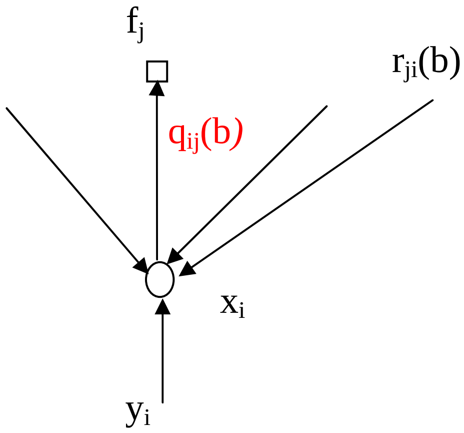
- C_i = {c-nodes connected to v-node i}
- $C_i \setminus j$ ={c-nodes connected to v-node i} \ {c-node j}

- P_i $= \Pr(c_i = 1 / y_i)$

Notation (cont'd)

- $q_{ij}(b)$ = message (extrinsic information) to be passed from node x_i to node f_j regarding the probability that $c_i = b, \quad b \varepsilon \{0,1\}$

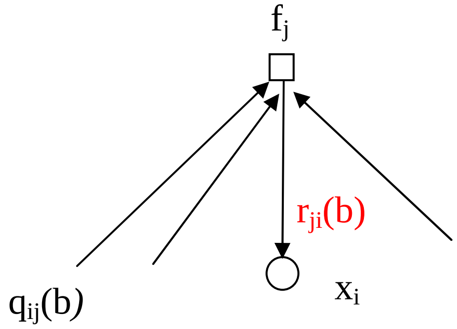
  = probability that $c_i = b$ given extrinsic information from all check nodes, except node f_j, and channel sample y_i



Notation (cont'd)

- $r_{ji}(b)$ = message to be passed from node f_j to node x_i

  = the probability of the j^{th} check equation being satisfied given bit c_i = b and the other bits have separable distribution given by $\{q_{ij'}\}_{j' \neq j}$



=========

- we now observe from Lemma 1 that

$$r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_j \setminus i} (1 - 2q_{i'j}(1))$$

$$r_{ji}(1) = \frac{1}{2} - \frac{1}{2} \prod_{i' \in V_j \setminus i} (1 - 2q_{i'j}(1))$$

- further, observe that, assuming independence of the { $r_{ji}(b)$},

Think of a repetition code.

$$q_{ij}(0) = (1 - P_i) \prod_{j' \in C_i \setminus j} r_{j'i}(0)$$

$$q_{ij}(1) = P_i \prod_{j' \in C_i \setminus j} r_{j'i}(1)$$

- as indicated above, the algorithm iterates back and forth between $\{q_{ij}\}$and $\{r_{ji}\}$; we already know how to pass the messages $q_{ij}(b)$ and $r_{ji}(b)$ around from the *Decoding Overview Section.*

- before we give the iterative decoding algorithm, we will need the following result

<u>Lemma 2</u> Suppose $y_i = x_i + n_i$, where $n_i \sim \eta(0,\sigma^2)$ and

$$\Pr(x_i = +1) = \Pr(x_i = -1) = {}^1\!/_2 .$$

Then $\Pr(x_i = x | y) \quad = \quad \dfrac{1}{1+e^{-2yx/\sigma^2}}$ (with $x \in \{\pm 1\}$)

<u>proof</u> (Bayes' rule and some algebra)

Sum-Product Algorithm Summary - Probability Domain

(perform looping below $\forall i, j$ for which $h_{ij} = 1$)

c_i=0

(0) initialize:

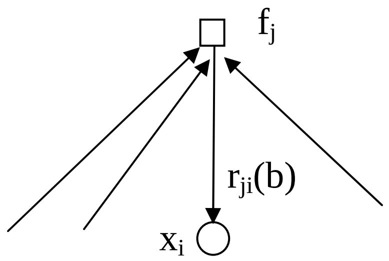
$$q_{ij}(0) = 1 - P_i = \Pr(x_i = +1 \mid y_i) \quad = \frac{1}{1+e^{-2y_i/\sigma^2}}$$

c_i=1

$$q_{ij}(1) = P_i = \Pr(x_i = -1 \mid y_i) \quad = \frac{1}{1+e^{2y_i/\sigma^2}}$$

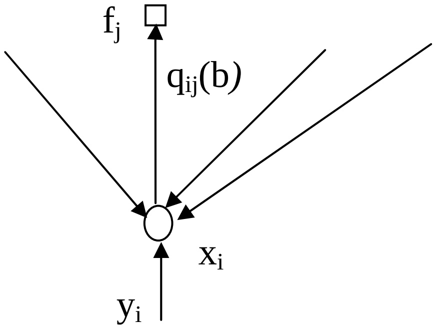
(1) $r_{ji}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in V_{j \backslash i}} (1 - 2q_{i'j}(1))$

$$r_{ji}(1) = 1 - r_{ji}(0)$$



Sum-Product Algorithm Summary (cont'd)

(2) $q_{ij}(0) = K_{ij}(1-P_i) \prod_{j' \in C_{i \setminus j}} r_{j'i}(0)$



$$q_{ij}(1) = K_{ij} P_i \prod_{j' \in C_{i \setminus j}} r_{j'i}(1)$$

where the constants K_{ij} are chosen to ensure $q_{ij}(0) + q_{ij}(1) \quad = \quad 1$

(3) Compute $\forall i$

$$Q_i(0) \quad = \quad K_i(1-P_i) \prod_{j \in C_i} r_{ji}(0)$$

$$Q_i(1) \quad = \quad K_i P_i \prod_{j \in C_i} r_{ji}(1)$$

where the constants K_i are chosen to ensure $Q_i(0) + Q_i(1) = 1$

Sum-Product Algorithm Summary (cont'd)

(4) • $\forall i$ $\quad \hat{c}_i = \begin{cases} 1 \text{ if } Q_i(1) > 0.5 \\ 0 \text{ else} \end{cases}$

- if $\left(\hat{\mathrm{c}}\mathrm{H}^{\mathrm{T}} = \overline{0}\right)$ OR $\left(\#\ \text{iterations} = \text{max\_iterations}\right)$ OR $(\text{other stopping rule})$

  then STOP

  else, go to (1)

=======

Remarks

(a) if the graph corresponding to the H matrix contains no cycles (i.e., is a “tree”), then $Q_i(0)$ and $Q_i(1)$ will converge to the true a posteriori probabilities for c_i as the number of iterations tends to ∞

(b) (for good LDPC codes) the algorithm is able to detect an “uncorrected” codeword with near-unity probability (step (4)), unlike turbo codes [MacKay]

(c) this algorithm is applicable to the binary symmetric channel where $q_{ij}(b) = \Pr(c_i = b \mid y_i), \quad b, y_i \in \{0,1\}$, still holds; we can also extend it to the binary erasure channel, fading channels, etc.

Log Domain Decoder

- as with the probability-domain Viterbi and BCJR algorithms, the probability-domain message-passing algorithm suffers because

  1) multiplications are involved (additions are less costly to implement)

  2) many multiplications of probabilities are involved which could become numerically unstable (imagine a very long code with 50-100 iterations)

- thus, as with the Viterbi and BCJR algorithms, a log-domain version of the message-passing algorithm is to be preferred

- to do so, we first define the following LLR’s:

$$c_i = 0$$

$$L(c_i) \equiv \log \frac{\Pr(x_i = +1 | y_i)}{\Pr(x_i = -1 | y_i)}$$

$$c_i = 1$$

$$L(r_{ji}) \equiv \log \frac{r_{ji}(0)}{r_{ji}(1)}$$

$$L(q_{ij}) \equiv \log \frac{q_{ij}(0)}{q_{ij}(1)}$$

$$L(Q_i) \equiv \log \frac{Q_{ij}(0)}{Q_{ij}(1)}$$

- we will also need the following result (trivial to show):

$$\begin{aligned} \tanh\left(\frac{1}{2}\log\left(p_0/p_1\right)\right) &= p_0 - p_1 \\ &= 1-2p_1 \end{aligned} \qquad (*)$$

- the initialization step thus becomes

$$\begin{aligned} L(q_{ij}) = L(c_i) &= \log \frac{\left(1+e^{-2y_i/\sigma^2}\right)^{-1}}{\left(1+e^{2y_i/\sigma^2}\right)^{-1}} \\ &= 2y_i/\sigma^2 \end{aligned}$$

- for step (1), we first rearrange the $r_{ji}(0)$ equation:

$$2r_{ji}(0) = 1 + \prod_{i' \in V_j \backslash i} \left(1 - 2q_{i'j}(1)\right)$$

$$\Rightarrow 1 - 2r_{ji}(1) = \prod_{i' \in V_j \backslash i} \left(1 - 2q_{i'j}(1)\right)$$

- from (*) on the previous page

$$\tanh\left(\frac{1}{2}L(r_{ji})\right) = \prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right) \qquad (**)$$

or

$$L(r_{ji}) = 2\tanh^{-1}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right)$$

- the problem with these expressions is that we are still left with a product

- we can remedy this as follows (Gallager):

  rewrite $L(q_{i'j})$ as

$$L(q_{i'j}) = \alpha_{ij} \beta_{ij}$$

  where

$$\begin{aligned} \alpha_{ij} &\equiv sign\big(L(q_{ij})\big) \\ \beta_{ij} &\equiv \big|L(q_{ij})\big| \end{aligned}$$

- then (**) on the previous page can be written as

$$\tanh\left(\frac{1}{2} L(r_{ij})\right) = \prod_{i'} \alpha_{ij} \cdot \prod_{i'} \tanh\left(\frac{1}{2} \beta_{i'j}\right)$$

- we then have

$$L(r_{ji}) = \left(\prod_{i'} \alpha_{ij}\right) \cdot 2\tanh^{-1} \underbrace{\qquad}_{\log^{-1}\log} \prod_{i'} \tanh\left(\frac{1}{2}\beta_{i'j}\right)$$

$$= \left(\prod_{i'} \alpha_{ij}\right) 2\tanh^{-1} \log^{-1} \sum_{i'} \log\tanh\left(\frac{1}{2}\beta_{i'j}\right)$$

$$= \left(\prod_{i'} \alpha_{ij}\right) \phi \left(\sum_{i' \in R_{j/i}} \phi(\beta_{i'j})\right)$$

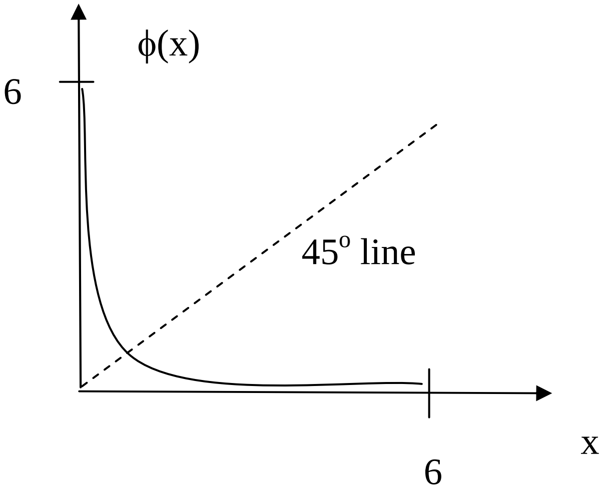
where we have defined

$$\phi(x) \equiv -\log\tanh\left(\frac{1}{2}x\right) = \log\frac{e^x+1}{e^x-1}$$

and used the fact that $\phi^{-1}(x) = \phi(x)$ when x > 0:

$$\phi\big(\phi(x)\big) = \log \frac{e^{\phi(x)} + 1}{e^{\phi(x)} - 1} = x$$

- the function $\phi(x)$ is fairly well behaved, it looks like



and so may be implemented via look-up table

- for step (2), we simply divide the $q_{ij}(0)$ eqn by the $q_{ij}(1)$ eqn and take the log of both sides to obtain

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i / j} L(r_{j'i})$$

- step (3) is similarly modified

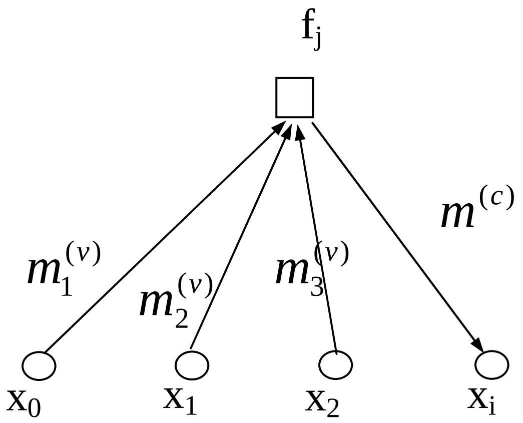
Sum-Product Algorithm Summary - Log Domain

(perform looping below $\forall i, j$ for which $h_{ij} = 1$)

(0) initialize:

$$L(q_{ij}) = L(c_i) \quad = \quad 2y_i / \sigma^2$$

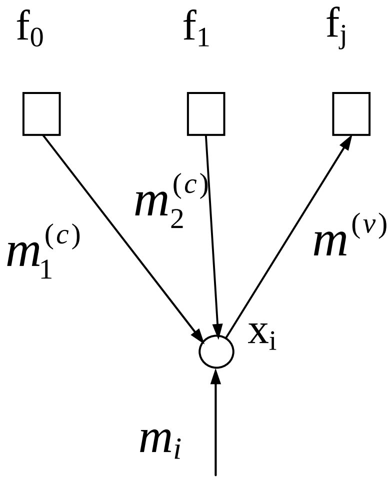
(1) $$L(r_{ji}) = \left(\prod_{i' \in V_{j \backslash i}} \alpha_{i'j} \right) \cdot \phi\left(\sum_{i' \in V_{j \backslash i}} \phi(\beta_{i'j}) \right)$$



where

$$\begin{aligned}
\alpha_{ij} &\equiv sign\left(L(q_{ij})\right) \\
\beta_{ij} &\equiv \left|L(q_{ij})\right| \\
\phi(x) &\equiv -\log\tanh(x/2) \\
&= \log\frac{e^x+1}{e^x-1}
\end{aligned}$$

(2) $$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$$



(3) $$L(Q_i) = L(c_i) + \sum_{j \in C_i} L(r_{ji})$$

(0) ● $\forall i$,

$$\hat{c}_i = \begin{cases} 1 & \text{if} \quad L(Q_i) < 0 \\ 0 & \text{else} \end{cases}$$

- if $\left(\hat{\mathrm{c}}\mathrm{H}^{\mathrm{T}} = \overline{0}\right)$ OR $\left(\#\ \text{iterations} = \text{max\_iterations}\right)$ OR (other stopping rule)

  then STOP

  else, go to (1)

==========

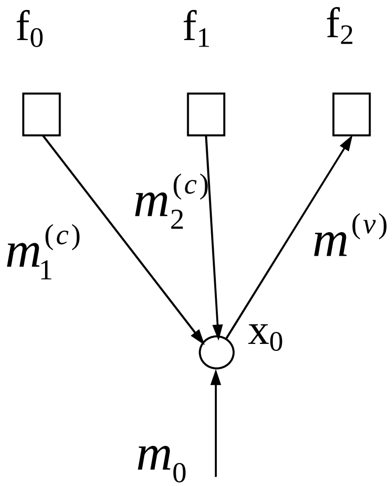
Improved Notation: Log-SPA Algorithm for LDPC Codes on the AWGN Channel

1. initialize the channel messages for each v-node via

$m_0 = 2y_i / \sigma^2$ (all other messages set to zero);

2. update the v-node messages via

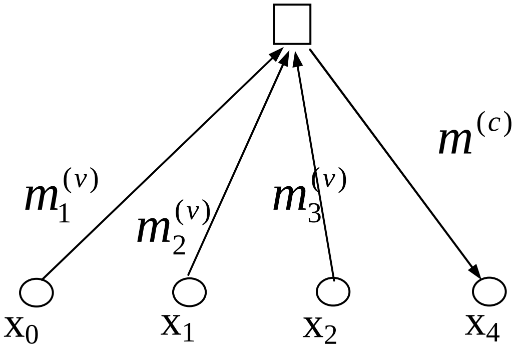
$$m^{(v)} = m_0 + \sum_{k=1}^{d_v-1} m_k^{(c)} ;$$



3. update the c-node messages via

$$m^{(c)} = \prod_k \alpha_k^{(v)} \cdot \phi\left(\sum_{k=1}^{d_c-1} \phi\left(\beta_k^{(v)}\right) \right);$$

where $\alpha_k^{(v)}$ and $\beta_k^{(v)}$ are the sign and magnitude of $m^{(v)}$



4. compute $M^{(v)} = m_0 + \sum_{k=1}^{d_v} m_k^{(c)}$ and $\hat{x} = sign\left(M^{(v)}\right)$ for each code bit to obtain $\hat{\mathbf{x}}$ (hence $\hat{\mathbf{c}}$);

5. if $\left(\hat{\mathrm{c}}\mathrm{H}^{\mathrm{T}} = \overline{0}\right)$ OR $(\#\,\text{iterations} = \text{max\_iterations})$ OR (other stopping rule), STOP, else go to step 2.

Comments

1. The order of the steps in the above algorithm summary is different than the summary given earlier in part to show that equivalent variations on the algorithm exist
2. In fact, the following ordering of the steps saves some computations (with a modification to Step 2):

   Step 1

   Step 4

   Step 5

   Step 2' This step is modified as: $m^{(v)} = M^{(v)} - m^{(c)}_{d_v}$

   (this is an extrinsic information calculation)

   Step 3 (and then go up to Step 4)

Example

- consider an (8,4) product code (d_{min}=4) composed of a (3, 2) single parity check code (d_{min}=2) along rows and columns:

c_0	c_1	c_2
c_3	c_4	c_5
c_6	c_7	

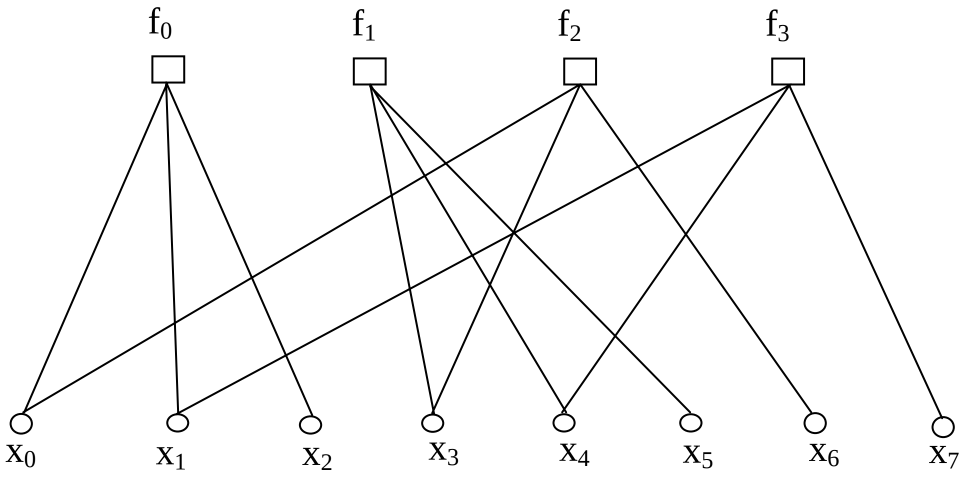
- thus,

$$
\begin{aligned}
c_2 &= c_0+c_1 \\
c_5 &= c_3+c_4 \\
c_6 &= c_0+c_3 \\
c_7 &= c_1+c_4
\end{aligned}
$$

from which

$$
H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}
$$

- the graph corresponding to H is



- note that the code is neither low-density nor regular, but it will suffice to demonstrate the decoding algorithm

- the codeword we choose for this example is

$$\bar{c} = \begin{array}{ccc} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & \end{array} \rightarrow \left(x_i = (-1)^{c_i}\right) \rightarrow \quad \bar{x} = \begin{array}{ccc} -1 & +1 & -1 \\ +1 & -1 & -1 \\ -1 & -1 & \end{array}$$

- the received word $\bar{y}$ is

$$\bar{y} = \begin{array}{ccc} y_0 & y_1 & y_2 \\ y_3 & y_4 & y_5 \\ y_6 & y_7 & \end{array} = \begin{array}{ccc} +.2 & +.2 & \_.9 \\ +.6 & +.5 & -1.1 \\ -.4 & -1.2 & \end{array}$$

so that there are sign errors in y_0 and y_4

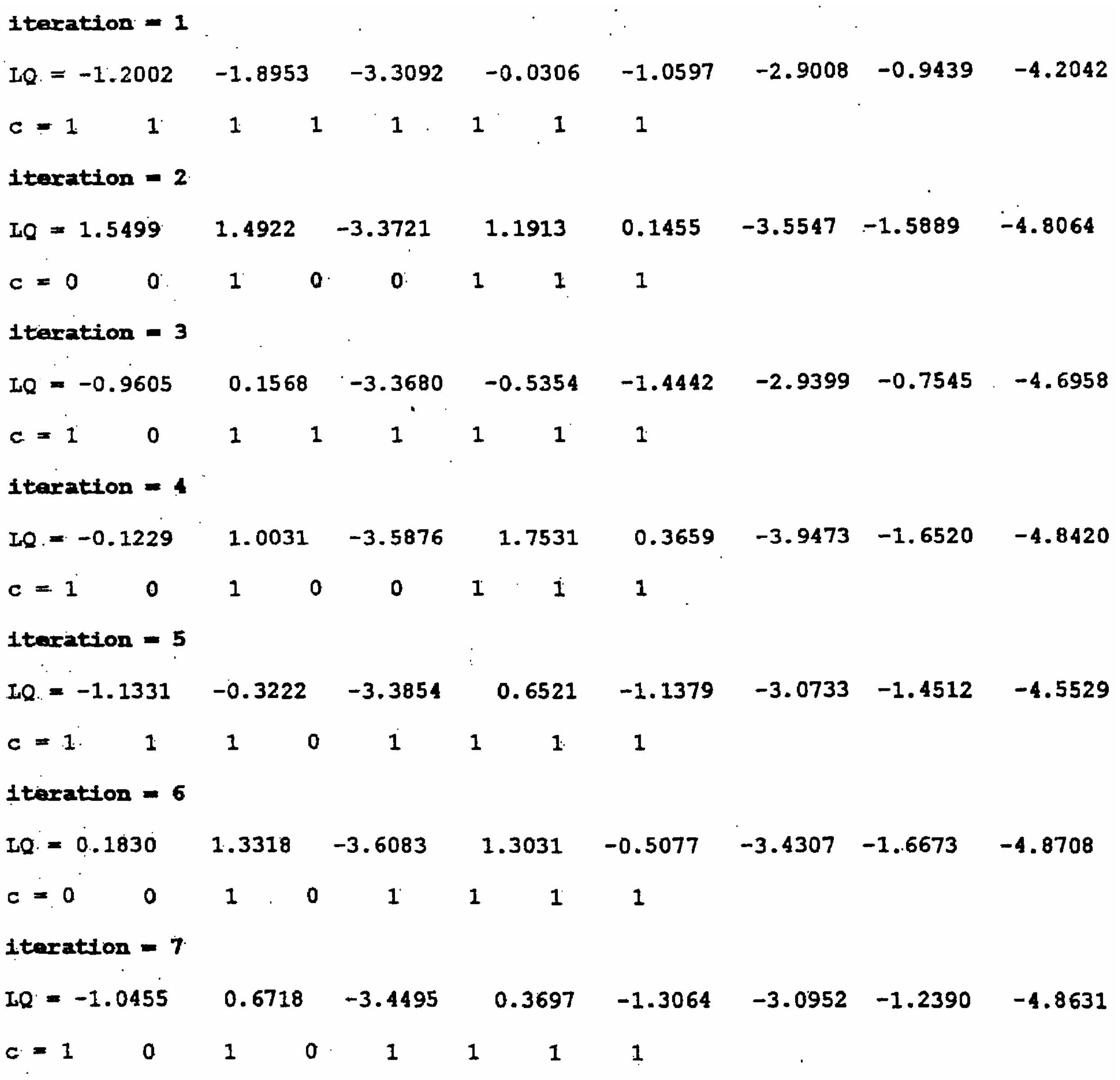
- initialization (assume σ^2=0.5):

$\forall i, j$ for which $h_{ij} = 1$:

set

$$L(q_{ij}) = L(c_i) \quad = \quad 2y_i / \sigma^2$$

we obtain $\left[LQ \equiv (L(Q_0),\ldots,L(Q_7))\right]$:



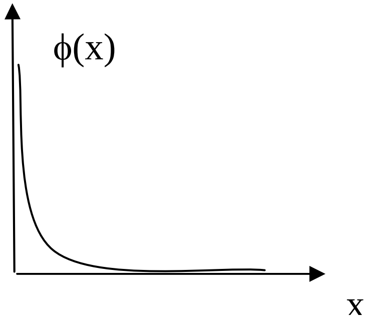
```
iteration = 1
LQ = -1.2002   -1.8953   -3.3092   -0.0306   -1.0597   -2.9008  -0.9439   -4.2042
c = 1     1     1     1     1     1     1     1
iteration = 2
LQ = 1.5499    1.4922   -3.3721    1.1913    0.1455   -3.5547  -1.5889   -4.8064
c = 0     0     1     0     0     1     1     1
iteration = 3
LQ = -0.9605    0.1568   -3.3680   -0.5354   -1.4442   -2.9399  -0.7545   -4.6958
c = 1     0     1     1     1     1     1     1
iteration = 4
LQ = -0.1229    1.0031   -3.5876    1.7531    0.3659   -3.9473  -1.6520   -4.8420
c = 1     0     1     0     0     1     1     1
iteration = 5
LQ = -1.1331   -0.3222   -3.3854    0.6521   -1.1379   -3.0733  -1.4512   -4.5529
c = 1     1     1     0     1     1     1     1
iteration = 6
LQ = 0.1830    1.3318   -3.6083    1.3031   -0.5077   -3.4307  -1.6673   -4.8708
c = 0     0     1     0     1     1     1     1
iteration = 7
LQ = -1.0455    0.6718   -3.4495    0.3697   -1.3064   -3.0952  -1.2390   -4.8631
c = 1     0     1     0     1     1     1     1
```

Reduced Complexity Decoder: The Min-Sum Algorithm

- consider the update equation for $L(r_{ji})$in the log-domain decoder:

$$L(r_{ji}) = \left(\prod_{i'} \alpha_{ij} \right) \cdot \phi\left(\sum_{i'} \phi(\beta_{i'j}) \right)$$

- notice now the shape of *ϕ(x):*



- we may conclude that the term corresponding to the smallest β_{ji} in the above summation dominates so that

$$\phi\left(\sum_{i'}\phi\left(\beta_{i'j}\right)\right) \simeq \phi\left(\phi\left(\min_{i'}\beta_{i'j}\right)\right)$$

$$= \min_{i'}\beta_{i'j}$$

- the second equality follows from $\phi\left(\phi(x)\right) = x$
- the min-sum algorithm is thus simply the log-domain algorithm with step (1) replaced by

$$(1') \qquad L(r_{ji}) = \prod_{i'\varepsilon R_{j\backslash i}} \alpha_{i'j} \cdot \min_{i'\varepsilon R_{j\backslash i}} \beta_{i'j}$$

- the min-sum algorithm converges in 10 iterations rather than 7 in the above example (since it is an approximation)
- to illustrate the simplicity of the min-sum algorithm, we change the received word in the example that follows.

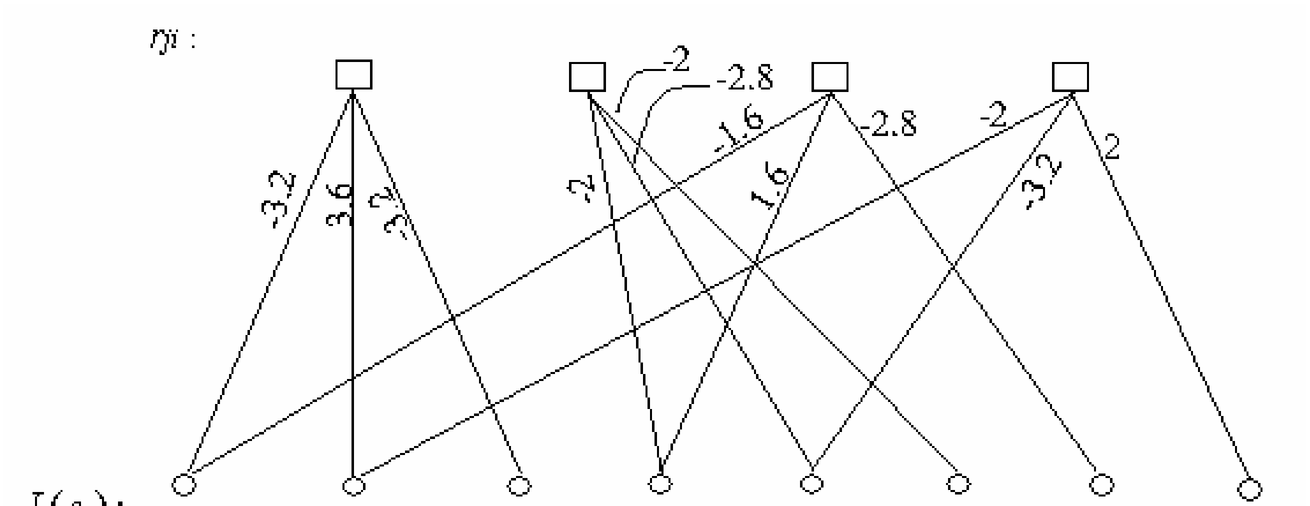
Example

$$\bar{y} = \begin{array}{rrr} -1.5 & .8 & -.9 \\ .7 & .5 & -1.1 \\ -.4 & -1.2 & \end{array}$$

- assuming again $\sigma^2 = 0.5$,

$$\begin{aligned} \{L(c_i)\}_{i=0}^{7} &= \left\{\frac{2y_i}{\sigma^2}\right\}_{i=0}^{7} \\ &= (-6, 3.2, -3.6, 2.8, 2, -4.4, -1.6, -4.8) \end{aligned}$$

- the worked-out graph, first iteration:



$L(c_i)$:	-6	3.2	-3.6	2.8	2	-4.4	-1.6	-4.8
$L(Q_i)$:	-10.8	4.8	-6.8	2.4	-4	-2.4	-4.4	-2.8
$\hat{c}_i$:	1	0	1	0	1	1	1	1

$$\hat{c}H^T = \overline{0} \qquad \Rightarrow \quad \text{STOP}$$

========

The Min-Sum-with-Correction-Term Algorithm

- In a code's Tanner graph, edges leading from bit nodes to check nodes indicate that the sum of those bit must equal zero

- In the Tanner-graph based decoder, we may consider the computation of extrinsic information departing a check node to be of the form

$$
\begin{aligned}
L(r_{ji}) = \quad & L\left(\sum_{i' \in V_{j\backslash i}} b_{i'} \right) \\
& = L\left(\cdots \oplus b_g \oplus b_h \oplus b_j \oplus b_k \oplus \cdots \right) \\
& \equiv \; \cdots \oplus L_g \oplus L_h \oplus L_j \oplus L_k \oplus \cdots
\end{aligned}
$$

(the operators for the last line should be $\boxplus$)

- Now we imagine that we want to compute $L(r_{ji})$ in serial fashion, accounting for the bits $\left(\cdots, b_g, b_h, b_j, b_k, \cdots\right)$ one at a time

- Thus, we would only need to perform pairwise computations of the form $L_1 \boxplus L_2$

- It can be shown that

$$L_1 \boxplus L_2 = \operatorname{sign}(L_1)\operatorname{sign}(L_2)\min(|L_1|, |L_2|) + s(L_1, L_2)$$

where

$$s(x,y) = \log\left(1 + e^{-|x+y|}\right) - \log\left(1 + e^{-|x-y|}\right)$$

- It can also be shown that s(x, y) can successfully approximated by

$$\tilde{s}(x,y) = \begin{cases} c & \text{if } |x+y| < 2 \text{ and } |x-y| > 2\,|x+y| \\ -c & \text{if } |x-y| < 2 \text{ and } |x+y| > 2\,|x-y| \\ 0 & \text{otherwise} \end{cases}$$

see W. E. Ryan, "An Introduction to LDPC Codes," in CRC Handbook for Coding and Signal Processing for Recording Systems (B. Vasic, ed.) CRC Press, to be published in 2004. (http://www.ece.arizona.edu/~ryan/)

$W_c = W_r = 64$

$c = 0.5$

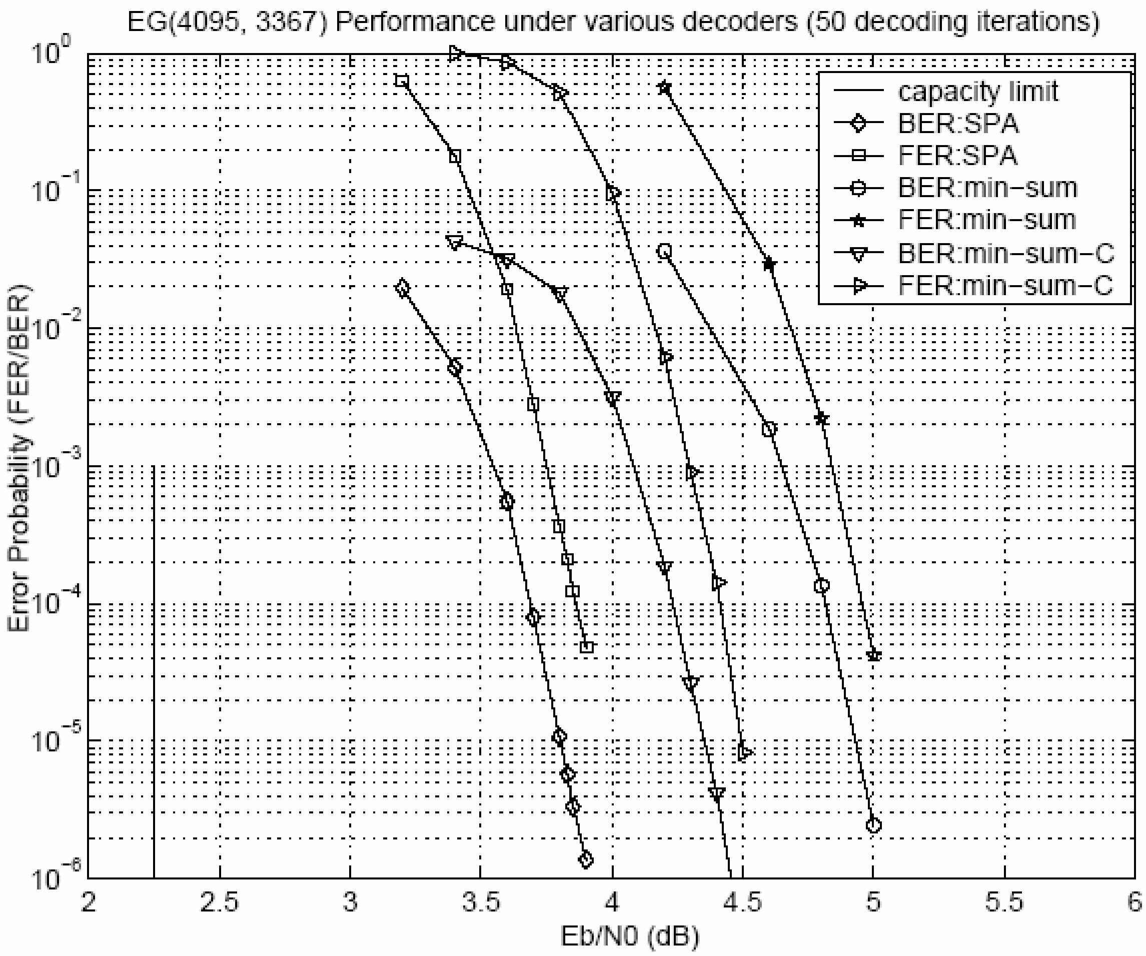


Fig. 8. Performance of a cyclic EG(4095, 3367) LDPC code on a binary-input AWGN channel and three decoding algorithms.

$W_c = 4$

$W_r = 32$

$c = 0.5$

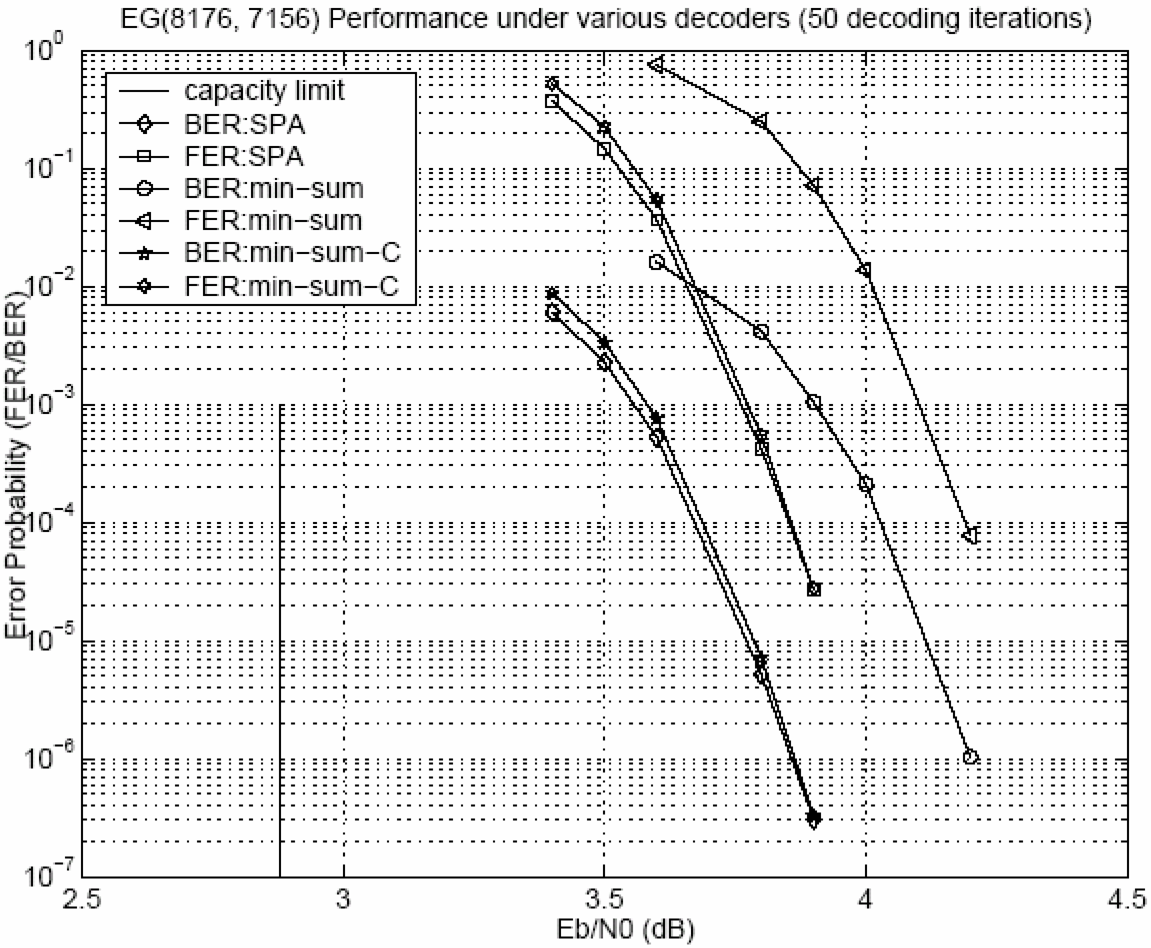
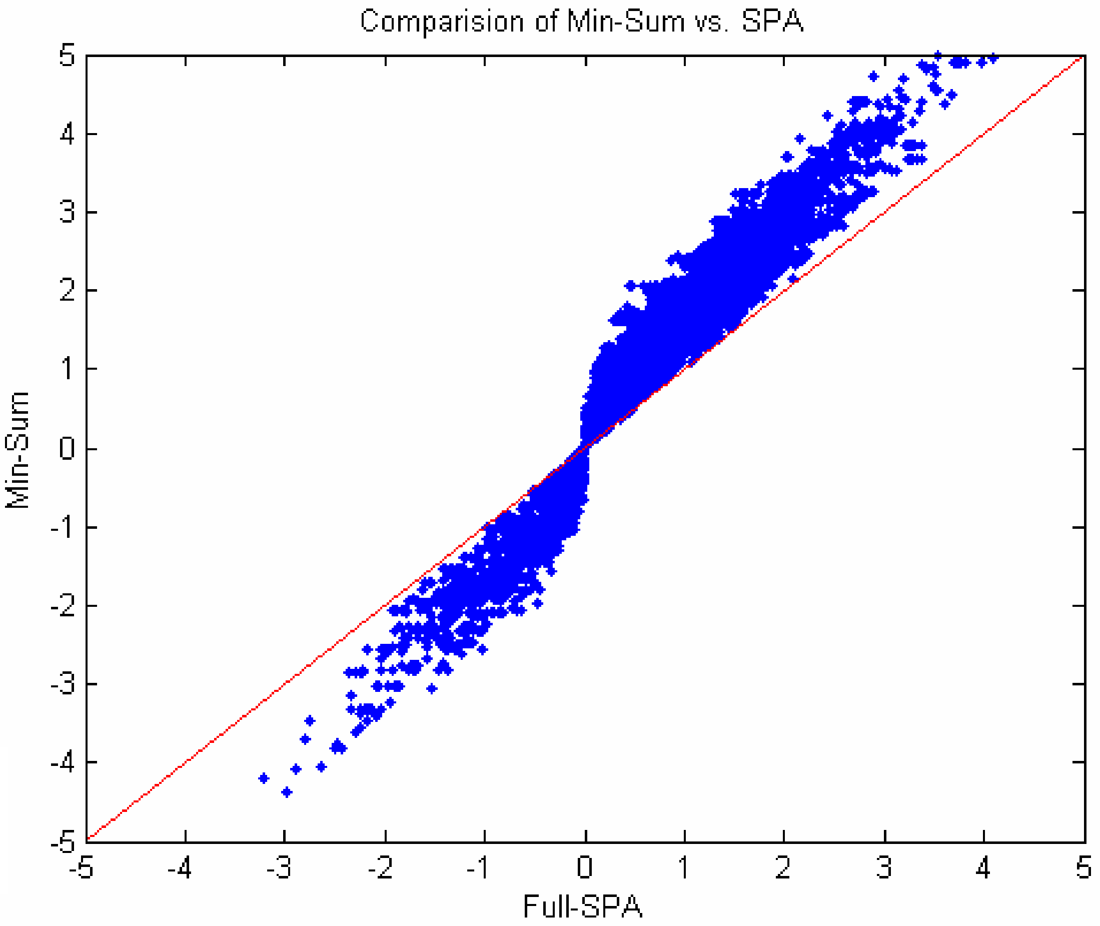


Fig. 9. Performance of a quasi-cyclic EG(8176, 7156) LDPC code on a binary-input AWGN channel and three decoding algorithms.

The Min-Sum-with-Attenuator Algorithm

- By plotting the min-sum LLR values against the "exact" SPA values (same data and noise), we see that the min-sum LLR values are generally too optimistic (too large, where large implies better reliability).



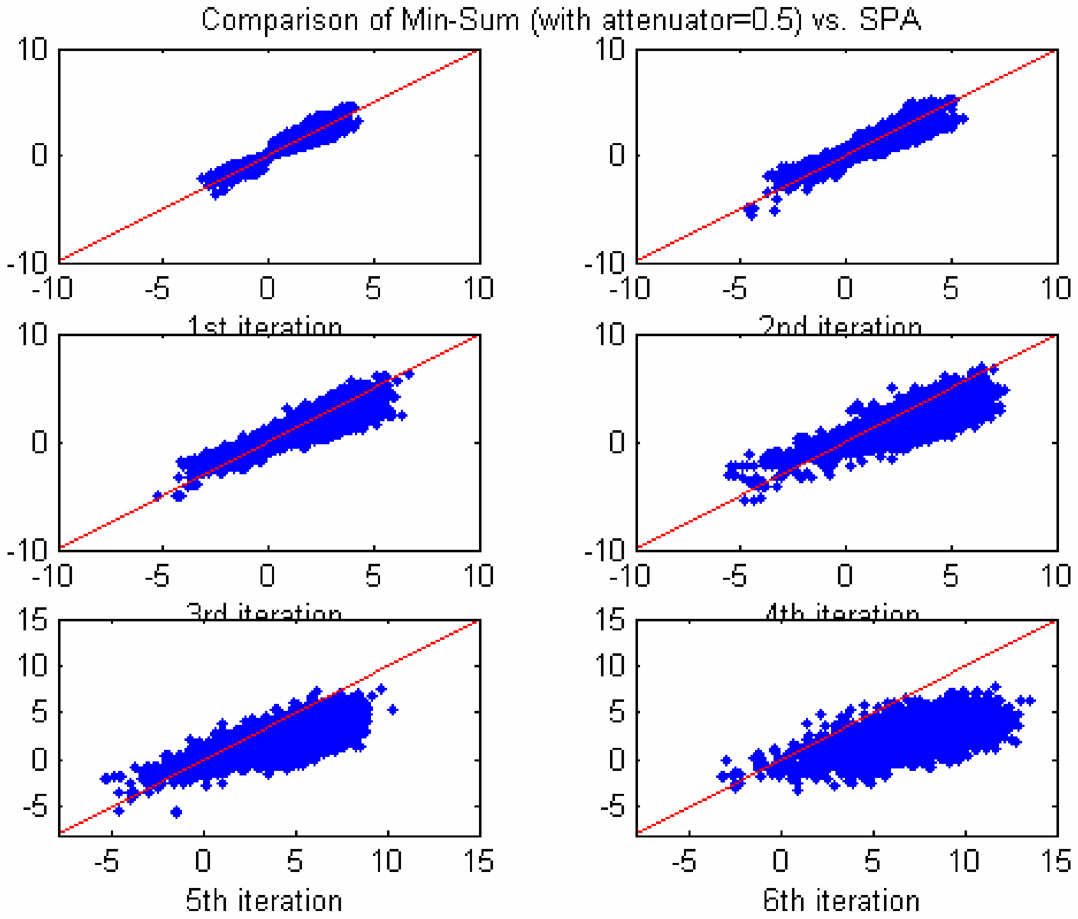
min-sum vs. SPA LLR's in the 1st iteration

- By attenuating the min-sum values, the resulting LLR values are sometimes optimistic and sometimes pessimistic (relative to the SPA), but a better balance is struck, hence the improvement over the min-sum algorithm.

new step (1):

$$(1'') \quad L(r_{ji}) = A \cdot \prod_{i' \varepsilon R_{j\backslash i}} \alpha_{i'j} \cdot \min_{i' \varepsilon R_{j\backslash i}} \beta_{i'j}$$

where 0 < A < 1 is the attenuation factor

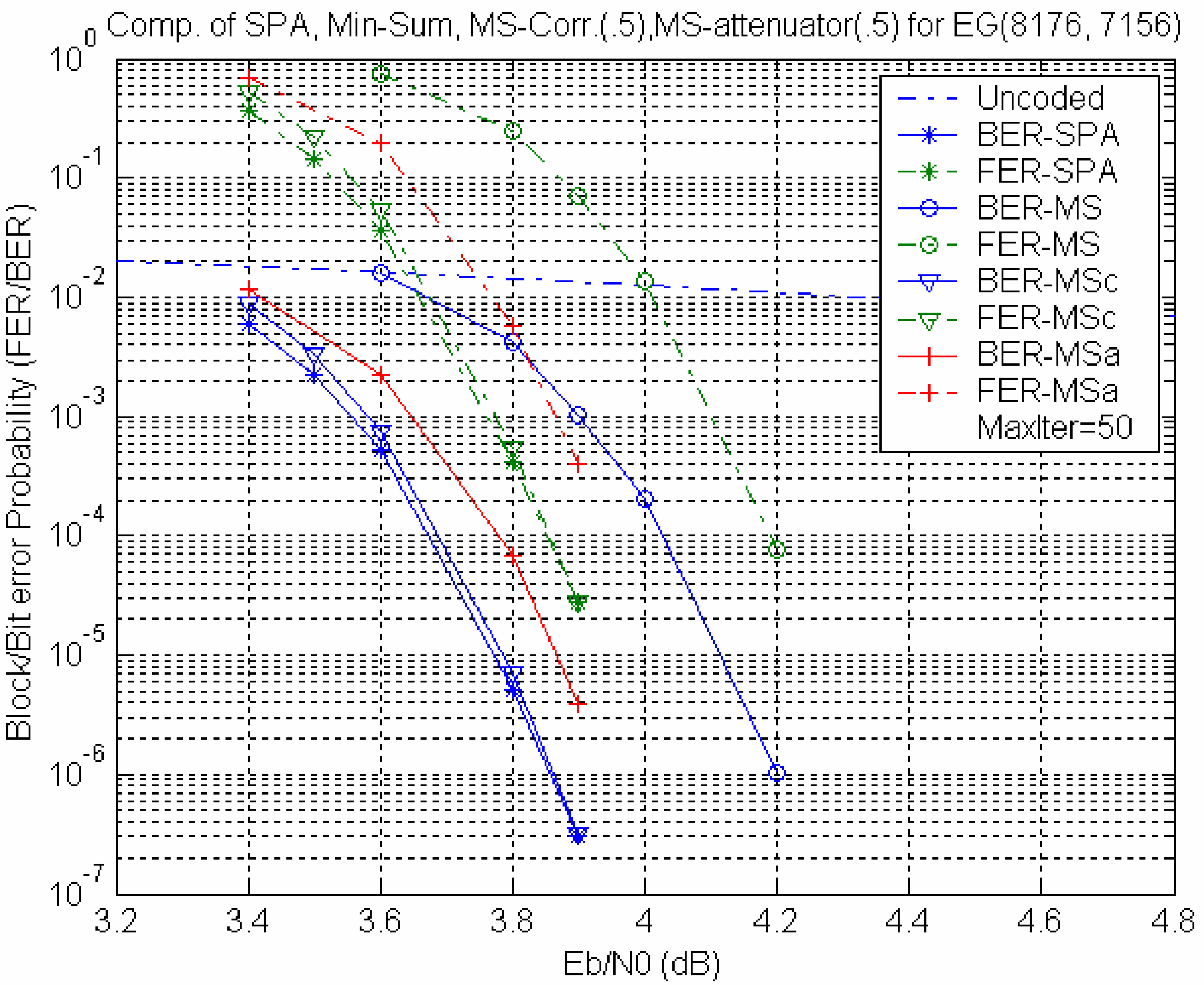


- It appears from the plot above that there might be some advantage to turning off the attenuation factor after 5 iterations because the values become generally pessimistic on the 6th iteration. We have not simulated this modification to the MSa.*x* algorithm.

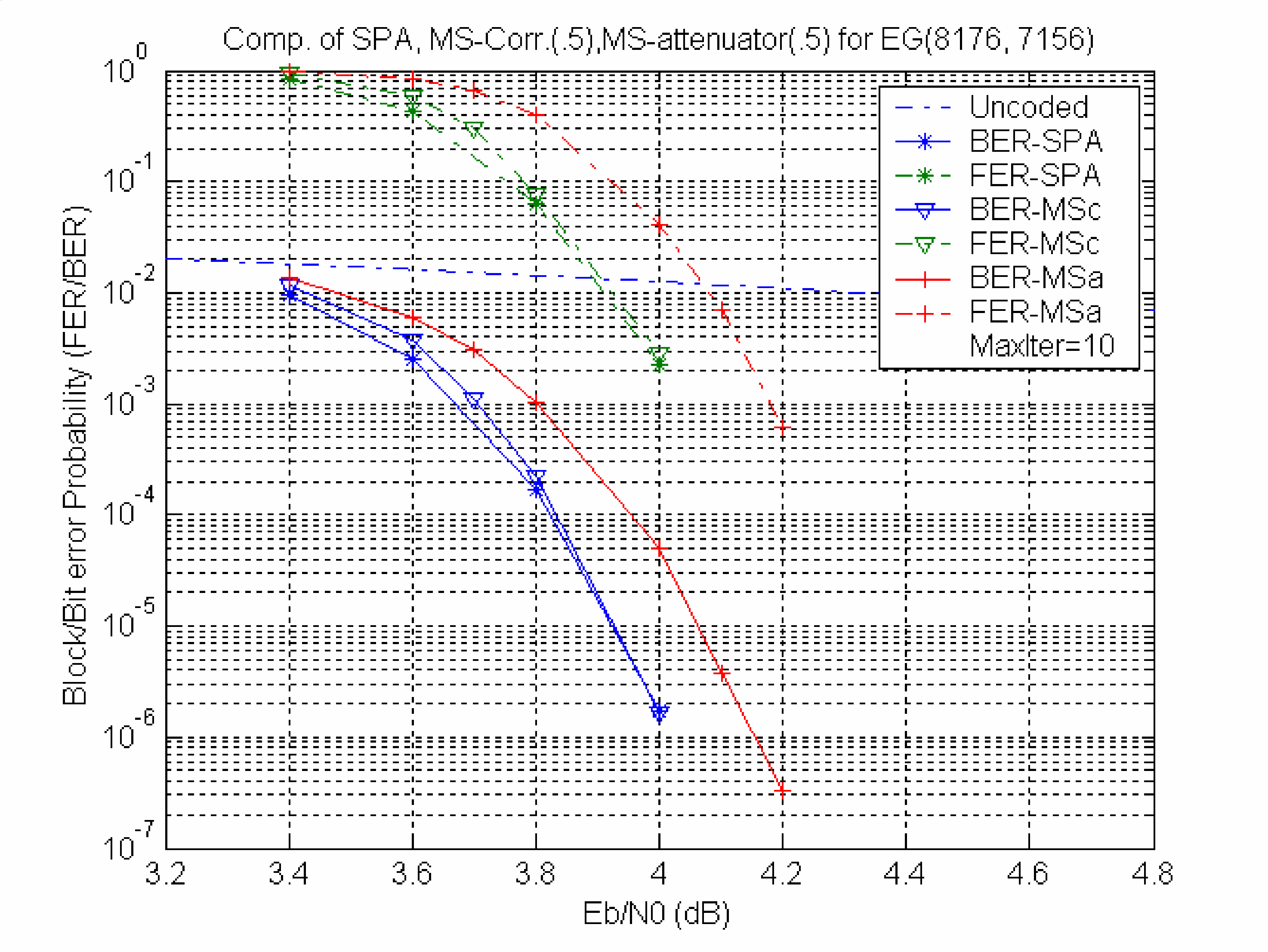
- Also, an attenuation factor of 0.8 is better than the factor of 0.5 shown here, but clearly the factor of 0.5 is to be preferred for practical reasons.

- Below we present some simulation results for the following algorithms:

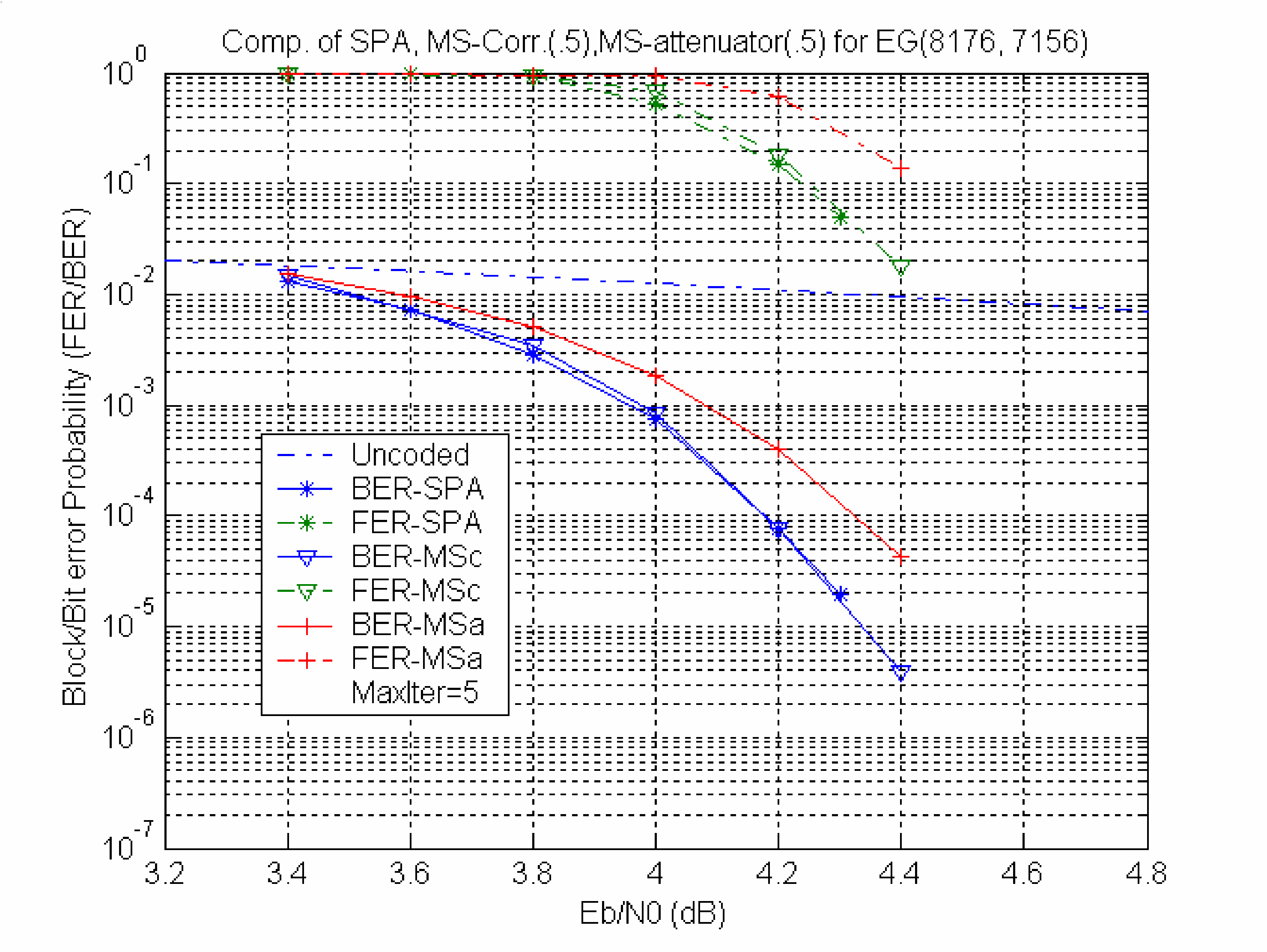
  - SPA = sum-product algorithm
  - MS = min-sum algorithm
  - MSc = min-sum with a correction term
  - MSa = min-sum with an attenuator



P_b and P_{cw} curves for MS, MSa.5(min-sum with attenuator 0.5), MSc.5 (min-sum with correction term 0.5), and SPA, all with **50 decoding iterations**.



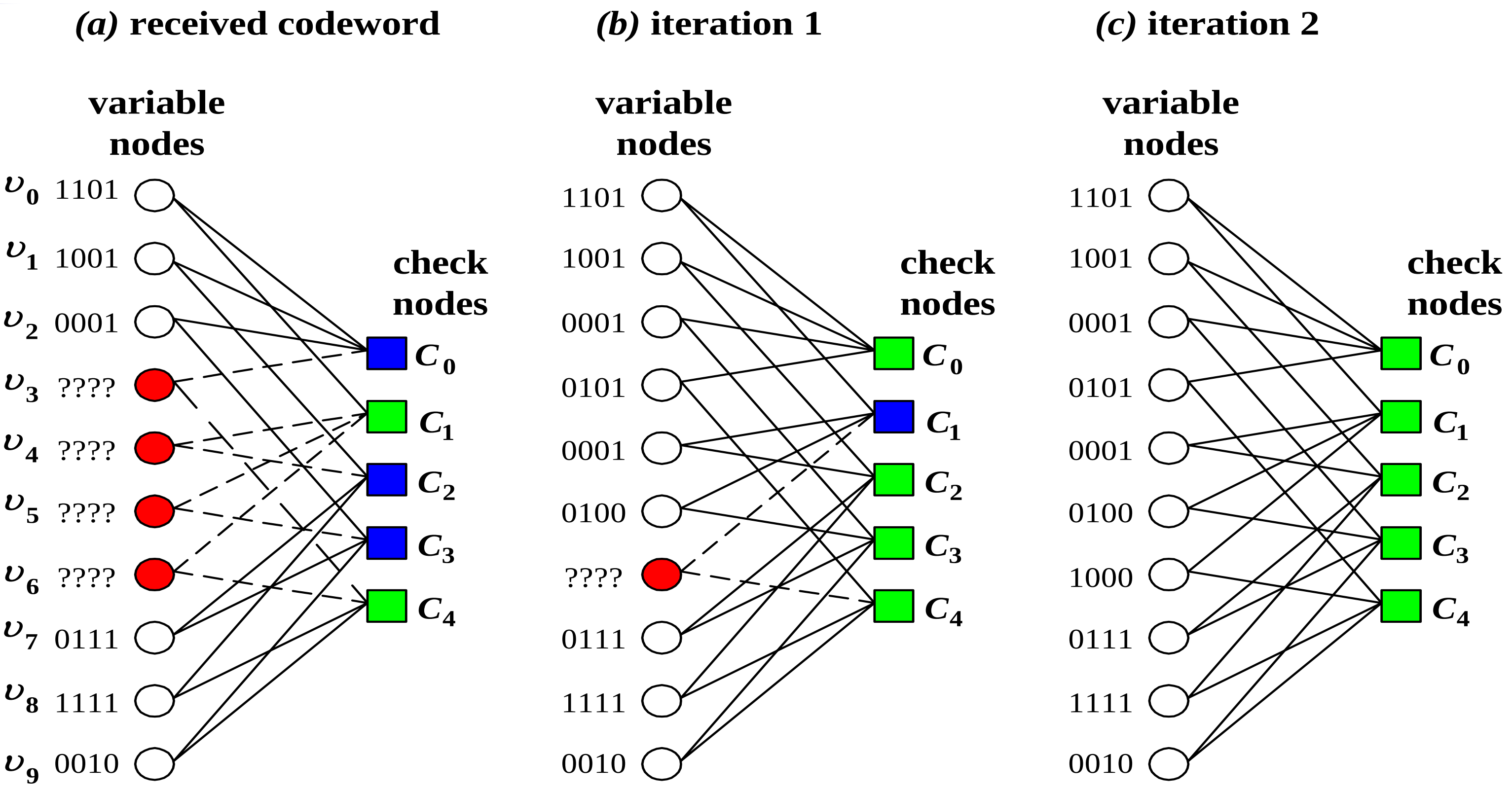
Repeat of the above, but with **10 iterations** and without the min-sum curves.



Repeat of the above, but with **5 iterations** and without the min-sum curves.

Iterative Decoding on the Packet Erasure Channel

An analogous algorithm of course applies for the binary erasure channel (BEC) or the burst erasure channel (BuEC).



REFERENCES (FOR PARTS I AND II)

Primary references used to write these notes:

R. Gallager, "Low-density parity-check codes," *IRE Trans. Information Theory*, pp. 21-28, Jan. 1962.

D. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Information Theory*, pp. 399431, March 1999.

J. Fan, *Constrained coding and soft iterative decoding for storage*, Ph.D. dissertation, Stanford University, December 1999. (See also his Kluwer monograph.)

Secondary references used to write these notes:

R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf Theory*, pp. 533-547, Sept. 1981.

J. Hagenauer, et al., "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf Theory*, pp. 429-445, March 1996.

M. Fossorier, et al. "Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation," *IEEE Trans. Comm*., pp. 673-680, May 1999.

T. Richardson, et al. "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf Theory*, Feb. 200 1.

D. MacKay, et al. "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Comm*., October 1999.

T. Richardson, et al. "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf Theory*, Feb. 2001.

M. Luby et al. "Improved low-density parity check codes using irregular graphs," *IEEE Trans. Inf. Theory*, Feb. 2001.

S.-Y. Chung, et al. "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Letters*, pp. 58-60, Feb. 2001.

Y. Kou, S. Lin, and M. Fossorier, "Low density parity check codes based on finite geometries: A rediscovery and more," submitted to *IEEE Trans. Inf. Theory*.

R. Lucas, M. Fossorier, Y. Kou, and S. Lin, "Iterative decoding of one-step majority-logic decodable codes based on belief propagation," *IEEE Trans. Comm*., pp. 931-937, June 2000.

M. Yang, W. E. Ryan and Y. Li, "Design of Efficiently Encodable Moderate-Length High-Rate Irregular LDPC Codes," *IEEE Trans. Comm*., April 2004.

W. E. Ryan, "An Introduction to LDPC Codes," in CRC Handbook for Coding and Signal Processing for Recording Systems (B. Vasic, ed.) CRC Press, to be published in 2004.
(http://www.ece.arizona.edu/~ryan/)